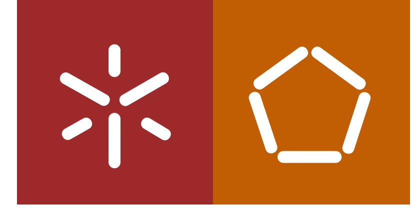
UMinho | 2016
Ricardo de Carvalho Oliveira Peixoto
Biblioteca e aplicação Android para comunicação
com câmaras ONVIF através de serviço REST

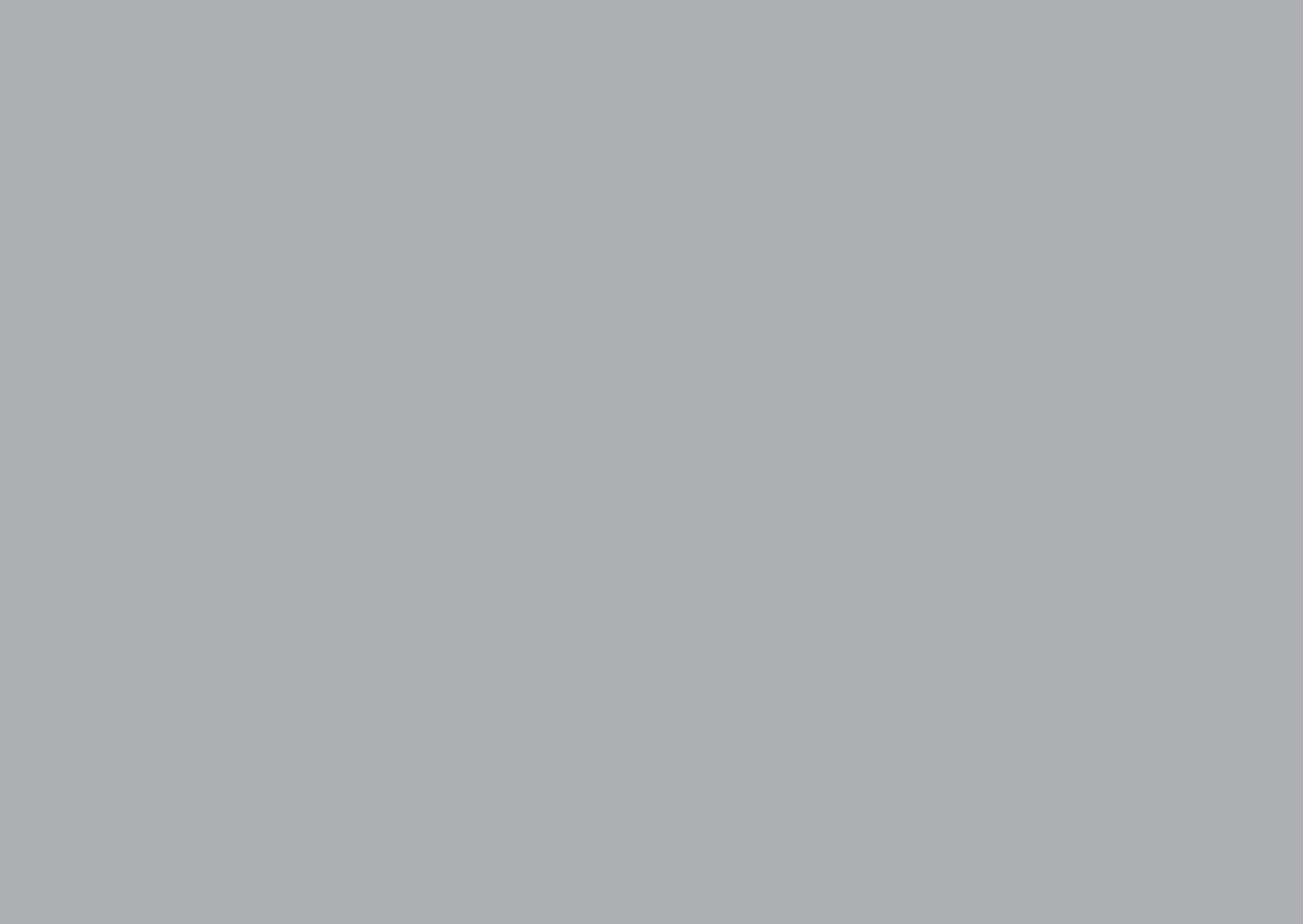


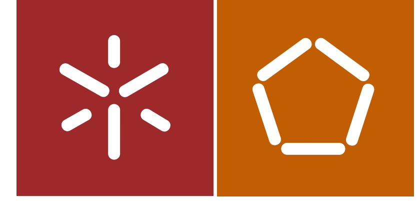
Universidade do Minho
Escola de Engenharia

Ricardo de Carvalho Oliveira Peixoto

Biblioteca e aplicação Android para comunicação com câmaras ONVIF através de serviço REST

outubro de 2016





Universidade do Minho
Escola de Engenharia

Ricardo de Carvalho Oliveira Peixoto

Biblioteca e aplicação Android para comunicação com câmaras ONVIF através de serviço REST

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de
Mestre em Engenharia de Telecomunicações e Informática

Trabalho efetuado sob a orientação do
Professor Doutor Sérgio Lopes

outubro de 2016

AGRADECIMENTOS

A elaboração deste trabalho deveu-se à colaboração, estimulo e empenho de diversas pessoas e instituições. Devido a este facto e chegado ao fim deste percurso gostaria de expressar toda a minha gratidão e apreço para todos aqueles contribuíram para que esta tarefa fosse possível. A todos, manifesto os meus sinceros agradecimentos.

À Universidade do Minho, pela oportunidade de fazer o curso.

Aos meus pais, pelo amor, apoio incondicional e incentivo em qualquer situação.

Obrigado meus pelos momentos partilhados, incentivo, partilha de experiência pela incontável contribuição.

Obrigado minha família que ao longo do meu percurso me motivou, deu apoio e pela contribuição valiosa.

Meus agradecimentos aos meus amigos que durante o meu percurso académico provaram ser excelentes companheiros de trabalho e verdadeiros irmãos na amizade, que, fizeram parte da minha formação e crescimento. Espero, com certeza, que irão continuar presentes na minha vida pessoal e profissional.

Ao Prof. Dr. Sérgio Adriano Fernandes Lopes pela disponibilidade oferecida, orientação, apoio e incansável suporte na realização desta dissertação.

A todos que direta ou indiretamente fizeram parte da minha formação.

O meu sincero obrigado a todos os que permitiram que esta dissertação fosse possível.

RESUMO

A diversidade de protocolos, formatos e especificidades de configuração utilizados pelos fabricantes de câmaras IP fez surgir a necessidade de normalizar a interação entre dispositivos. Nesse contexto surgiu o ONVIF, a norma para comunicação entre dispositivos multimédia mais adotada no mercado. O ONVIF é na atualidade a norma mais utilizada por dispositivos de vigilância em rede.

O Android é o sistema operativo mais utilizado em *smartphones* e *tablets*, e estes dispositivos são práticos para fazer a configuração de dispositivos. No entanto, o ONVIF é baseado em normas de *web services* SOAP, o que o torna bastante pesado em termos computacionais, sobretudo para dispositivos móveis. Um dos aspetos mais críticos é a serialização e desserialização de dados XML. As aplicações móveis atuais não se destacam em termos de desempenho da comunicação, tornando-se necessária a criação de soluções melhores. A filosofia REST é computacionalmente muito mais leve, porque ele usa formatos de dados simples e compacto diretamente através de HTTP.

Este trabalho pretende explorar as potencialidades de UI (*user-interface*) da plataforma. A aplicação serve para visualizar conteúdos multimédia e lidar com configurações das câmaras IP. A plataforma Android suporta diversos formatos de vídeo, áudio e disponibiliza ferramentas de UI para interagir com utilizador. A utilidade desta aplicação pode ir desde a simples configuração e acesso a câmaras ONVIF, passando pela demonstração/teste de funcionalidades ONVIF, até à monitorização remota de espaços.

Os principais componentes do trabalho são o serviço *web* existente, biblioteca Java REST ONVIF e aplicação Android para comunicação com a câmaras através da biblioteca.

Atualizou-se as configurações do servidor do serviço REST para a versão 2.4 do Apache, a mais recente à data. O serviço teve adições de funcionalidade para obter os serviços suportados e as *capabilities* das câmaras. Estas funcionalidades foram fundamentais na implementação da biblioteca de modo a fornecer acesso aos serviços e operações suportadas pela câmara.

No desenho da biblioteca e aplicação recorreu-se a diagramas de classes e de sequência para otimizar a solução. Este foram essenciais no desenho e foram a base da implementação destes componentes.

Desenvolveu-se a biblioteca Java REST ONVIF que implementa uma abstração das funções do serviço *web.* Conseguiu-se implementar uma solução que suporta a integração de modos de

comunicação futuros sem necessidade de reestruturação. Testou-se os métodos da biblioteca que apresentaram resultados funcionais.

A aplicação Android implementa uma estrutura de navegação por câmaras e por serviços de um NVT. Esta aproveita as funcionalidades de duas aplicações existentes e integra a comunicação REST através da biblioteca desenvolvida. Através da aplicação testou-se funcionalidades da biblioteca apresentando resultados perfeitamente funcionais.

ABSTRACT

The diversity of protocols, formats and specifications used by IP camera manufacturers raised the need to standardize the interaction between devices. In this context the ONVIF came, the standard for communication between most adopted multimedia devices on the market. Currently, the ONVIF standard is the most widely used for network monitoring devices.

Android is the most used operating system in smartphones and tablets, and these devices are practical to make the device configuration. However, ONVIF is based on SOAP web services standards, which makes it quite heavy in computational terms, especially for mobile devices. One of the most critical aspects is the serialization and de-serialization of XML data. Current mobile applications do not stand out in terms of communication performance, making it necessary to create better solutions. The REST philosophy is computationally much lighter because it uses simple data formats and compact directly over HTTP.

This work aims to explore the potential UI (user-interface) platform. The application should be used to view multimedia content and handle configurations of the IP cameras. The Android platform supports multiple video formats, audio and provides UI tools to interact with the user. The usefulness of this application goes from single configuration and access to ONVIF cameras, through demonstration/test ONVIF functionality, to remote monitoring spaces.

The main components in this work are the existing web service, Java REST ONVIF library and Android application for communication with the cameras through the library.

The service had functionality additions for supported services and capabilities of the cameras. These features are critical in the implementation of the library to provide access to services and operations supported by the camera.

In the library design and application appealed to class and sequence diagrams to optimize the solution. This was essential in the solution design and were the basis of the implementation of these components.

It was developed the Java REST ONVIF library that implements an abstraction of the web service functions. It managed to implement a solution that supports the integration of future communication modes without the need for restructuring. The library methods presented functional results.

The Android application implements a navigation structure for cameras and services of an NVT. This takes advantage of the features of two existing applications and integrates REST communication through the developed library. The application's features showed functional outcomes.

ÍNDICE

Agradecimentos v

Resumo vii

Abstract ix

Lista de Figuras xv

Lista de Tabelas xxi

Lista de Abreviaturas, Siglas e Acrónimos xxiii

1. Introdução. 1
   - 1.1 Enquadramento 1
   - 1.2 Motivação 2
   - 1.3 Objetivos 3
   - 1.4 Estrutura da dissertação 4
2. Estado da arte 7
   - 2.1 ONVIF 7
     - 2.1.1 Tipos de dispositivos 7
     - 2.1.2 Serviços de dispositivos NVT 8
     - 2.1.3 Resumo dos Serviços 8
   - 2.2 Serviços REST para ONVIF 9
     - 2.2.1 Serviços *web* REST 9
     - 2.2.2 Bibliotecas Cliente ONVIF 11
     - 2.2.3 Arquitetura do software 13
     - 2.2.4 Arquitetura do serviço REST existente 15
   - 2.3 Aplicações Android para ONVIF 16
     - 2.3.1 ONVIF IP Camera Monitor 16
     - 2.3.2 tinyCam Monitor FREE 20
     - 2.3.3 IP Cam Viewer Basic 22
     - 2.3.4 Sumário 24
     - 2.3.5 Aplicações base 25
3. Servidor REST ONVIF 27

3.1 Estrutura de diretorias do Apache ... 27
3.2 Processo de configuração ... 29
3.3 Renovação da configuração do serviço ... 33
3.4 Funcionalidades adicionadas ... 34
4. Biblioteca Java REST ONVIF ... 37
4.1 Requisitos ... 37
4.2 Desenho da API ... 38
4.2.1 Classe OnvifCamera ... 39
4.2.2 Classes de serviço ... 40
4.2.3 Classes de dados ... 45
4.2.4 Serialização e desserialização JSON ... 46
4.2.5 Desempenho ... 46
4.2.6 Análise de exceções ... 47
4.3 Implementação ... 48
4.3.1 Classes de utilidades ... 50
4.3.2 Classe OnvifHttpREST ... 51
4.3.3 Serialização e desserialização JSON ... 56
4.3.4 Classes de suporte a métodos assíncronos ... 58
4.3.5 Classes de dados ... 59
4.3.6 Classe OnvifCamera ... 60
4.3.7 Classes de serviço ... 61
4.3.8 Métodos síncronos ... 64
4.3.9 Métodos assíncronos ... 64
4.3.10 Informação de Progresso ... 66
4.3.11 Listeners disponibilizados ... 71
4.4 Utilização da biblioteca ... 72
5. Aplicação Android ... 75
5.1 Requisitos ... 75
5.2 Aplicações de base revisitadas ... 75
5.2.1 Aplicação com suporte para alguns serviços ... 76

5.2.2 Aplicação para Media ... 77

5.3 Análise ... 78

5.3.1 Justificação da escolha ... 79

5.3.2 Nova Aplicação ... 79

5.4 Desenho da Aplicação ... 80

5.5 Implementação ... 83

5.5.1 Estrutura da GUI ... 84

5.5.2 Interface de input e output ... 86

5.5.3 Estado de rede ... 87

5.5.4 Sistema de armazenamento de dados ... 87

5.5.5 Base de dados de câmaras ... 88

5.5.6 Gestão de memória ... 88

5.5.7 Toolbar ... 90

5.5.8 Otimização no uso de recursos ... 91

5.5.9 Normalizações para os menus de serviço ... 91

5.5.10 Fragmentos do serviço Media e Imaging ... 93

6. Conclusão ... 95

6.1 Resultados ... 95

6.2 Conclusões ... 96

6.3 Trabalho Futuro ... 97

Bibliografia ... 99

Anexo I – Configurações do Servidor Web REST ... 105

Anexo II – Plataforma Android ... 107

Anexo III – Funções suportadas pelo Serviço web REST ... 113

Anexo IV – Fundamentos JSON ... 119

Anexo V – Classes de Dados ... 121

Anexo VI – Classes de Serviço ... 143

Anexo VII – Análise trafego ... 149

Anexo VIII – Processo instalação servidor web REST ... 151

Anexo IX - Resultados da análise do tipo de progresso ... 157

Anexo X – Funcionalidades e Menus da nova aplicação 159

LISTA DE FIGURAS

Figura 1.1 - Arquitetura Geral do Sistema ... 3
Figura 2.1 - Mapeamento REST exemplo [5]. ... 11
Figura 2.2 - Arquitetura do servidor REST ONVIF ... 14
Figura 2.3 - Diagrama temporal de um pedido ao servidor ... 14
Figura 2.4 - Menu para adicionar uma câmara ... 17
Figura 2.5 - Menu inicial com as câmaras disponíveis ... 17
Figura 2.6 - Conteúdo do HTTP header ... 18
Figura 2.7 - Dados HTTP do pedido GetSystemDateAndTime ... 18
Figura 2.8 - Tamanho em bytes do pedido GetSystemDateAndTime ... 18
Figura 2.9 - Menu inicial com câmara configurada ... 21
Figura 2.10 - Navigation Drawer do menu inicial ... 21
Figura 2.11 - Menu principal com uma câmara configurada ... 23
Figura 3.1 - Sistema de ficheiros do apache2 ... 28
Figura 3.2 - Documentação e conteúdo do pedido getSystemDateAndTime ... 36
Figura 4.1 - Componentes principais da biblioteca Java REST ONVIF ... 38
Figura 4.2 - Diagrama de classes para dois modos de comunicação ... 40
Figura 4.3 – Diagramas de classes de serviço ... 41
Figura 4.4 - Diagrama de packages da Biblioteca REST ONVIF ... 49
Figura 4.5 - Diagrama de classes do package org.uminho.onvif. ... 50
Figura 4.6 - Diagrama da classe OnvifUtilities ... 51
Figura 4.7 - Classe HttpRest para comunicação HTTP ... 52
Figura 4.8 - Diagrama de sequência de um pedido get ... 54
Figura 4.9 - Diagrama de sequência do pedido post ... 55
Figura 4.10 - Equivalência entre um objeto dados e um objeto JSON para uma coleção de scopes . 57
Figura 4.11 - Diagrama das classes OnvifAsyncTask e OnvifAsyncTaskNoResult ... 58
Figura 4.12 - Diagrama de classes para a OnvifCamera ... 60
Figura 4.13 - Diagrama de classes para a relação dos serviços com os listeners e AsyncTakss ... 62
Figura 4.14 - Diagrama da classe abstrata OnvifService ... 63

Figura 4.15 - Diagrama de classes de serviço do Device IO 63
Figura 4.16 - Diagrama sequência do template dos métodos assíncronos com retorno 65
Figura 4.17 - Diagrama sequência do template dos métodos assíncronos sem retorno 65
Figura 4.18 – Implementação do método getSnapshotURI, versão assíncrona. 66
Figura 4.19 - Médias temporais do pedido GetMediaProfiles... 68
Figura 4.20 - Médias temporais do pedido SetSystemDateAndtime 69
Figura 4.21 - Diagrama de sequência para a solução final do progresso da biblioteca 70
Figura 4.22 - Diagrama de classes dos listeners... 72
Figura 4.23 - Criação de um objeto da OnvifCamera 73
Figura 4.24 - Acesso a um objeto de serviço da OnvifCamera... 73
Figura 4.25 - Verificar se serviço é suportado pela câmara 73
Figura 4.26 - Criação dos objetos de serviço Device Managment e Imaging 73
Figura 4.27 - Criação de um objeto OnvifErrorListener 74
Figura 4.28 - Invocação de um método síncrono com retorno 74
Figura 4.29 - Invocação de um método síncrono sem retorno.. 74
Figura 4.30 - Invocação de um método assíncrono com retorno 74
Figura 4.31 - Invocação de um método assíncrono sem retorno 74
Figura 5.1 – Arquitetura da aplicação.. 80
Figura 5.2 – Estrutura da GUI 81
Figura 5.3 - Estrutura das atividades gestão de câmaras e dos serviços 84
Figura 5.4 - Estrutura para fragmentos de serviço com tabuladores 85
Figura 5.5 - Modelo de base de dados para câmaras 88
Figura 5.6 - Memória libertada e alocada pela aplicação sem efetuar pedidos 89
Figura 5.7 - Memória libertada e alocada pela aplicação depois de efetuar diversos pedidos 89
Figura 5.8 - Toolbar do serviço Imaging... 90
Figura 5.9 - Toolbar da atividade de gestão de câmaras.. 90
Figura II.0.1 – Ciclo de vida da Activity [64] 108
Figura II.0.2 - Ciclo de vida de um Fragment enquanto a sua activity está aberta [57] 110
Figura III.0.1 - Diagrama do serviço Device Managment... 113
Figura III.0.2 - Diagrama do serviço Media.. 114
Figura III.0.3 - Diagrama do serviço DeviceIO 115

Figura III.0.4 - Diagrama do serviço Imaging 115
Figura III.0.5 - Diagrama do serviço PTZ 116
Figura III.0.6 - Diagrama do serviço Events 117
Figura IV.0.1 - Formato de um object JSON [73] 119
Figura IV.0.2 - Formato de um array JSON [73] 119
Figura IV.0.3 - Formato do *value* JSON [73] 119
Figura V.0.1 - Diagrama de classes de dados do serviço DeviceIO 121
Figura V.0.2 - Diagrama de classes de dados para as capabilities do DeviceMgmt 122
Figura V.0.3 - Diagrama de classes de dados para a AccountInformation do Device Managment ... 122
Figura V.0.4 - Diagrama de classes de dados para a DynamicDNSInformation do Device Managment 122
Figura V.0.5 - Diagrama de classes de dados para a DateTimeInformation do Device Managment 122
Figura V.0.6 . Diagrama de classes de dados para a ScopeInformation do Device Managment 123
Figura V.0.7 - Diagrama de classes de dados para a DNSInformation e DNSSettings do Device Managment 123
Figura V.0.8 - Diagrama de classes de dados para a DeviceInformation, Service e FactoryDefaultType do Device Managment 123
Figura V.0.9 - Diagrama de classes de dados para a NTPInformation do Device Managment 124
Figura V.0.10 - Diagrama de classes de dados para a NVTServices do Device Managment 124
Figura V.0.11 - Diagrama de classes de dados de dados do serviço Discovery 124
Figura V.0.12 - Diagrama de classes de dados do serviço Events 125
Figura V.0.13 - Diagrama de classes de dados para as capabilities do serviço Imaging 125
Figura V.0.14 - Diagrama de classes de dados para os movimentos de focus do serviço Imaging . 125
Figura V.0.15 - Diagrama de classes de dados para as opções de movimentos do serviço Imaging 126
Figura V.0.16 - Diagrama de classes de dados para o estado de imagem do serviço Imaging 126
Figura V.0.17 - Diagrama de classes de dados das definições de imagem para o serviço Imaging 127
Figura V.0.18 - Diagrama de classes de dados para white balance information do serviço Imaging 127
Figura V.0.19 - Diagrama de classes de dados para wide dynamic range information do serviço Imaging 127
Figura V.0.20 - Diagrama de classes de dados para focus configuration do serviço Imaging 128

Figura V.0.21 - Diagrama de classes de dados para back light compensation do serviço Imaging . 128
Figura V.0.22 - Diagrama de classes de dados para exposure settings do serviço Imaging 128
Figura V.0.23 - Diagrama de classes de dados para image options do serviço Imaging 129
Figura V.0.24 - Diagrama de classes de dados para white balance options do serviço Imaging 129
Figura V.0.25 - Diagrama de classes de dados para exposure options do serviço Imging 129
Figura V.0.26 - Diagrama de classes de dados para focus options do serviço Imging 130
Figura V.0.27 - Diagrama de classes de dados para wide dynamic range options do serviço Imaging 130
Figura V.0.28 - Diagrama de classes de dados para back light compensation options do serviço Imaging 130
Figura V.0.29 - Diagrama de classes de dados para media capabilities do serviço Media 131
Figura V.0.30 - Diagrama de classes de dados para codecs do serviço Media 131
Figura V.0.31 - Diagrama de classes de dados para media options do serviço Media 132
Figura V.0.32 - Diagrama de classes de dados para video encoder configuration options do serviço Media 132
Figura V.0.33 - Diagrama de classes de dados para JPEG options do serviço Media 132
Figura V.0.34 - Diagrama de classes de dados para MPEG4 options do serviço Media 133
Figura V.0.35 - Diagrama de classes de dados para H264 options do serviço Media 133
Figura V.0.36 - Diagrama de classes de dados para video source configuration options do serviço Media 133
Figura V.0.37 - Diagrama de classes de dados para audio encoder configuration options do serviço Media 134
Figura V.0.38 - Diagrama de classes de dados para PTZ configuration options do serviço Media .. 134
Figura V.0.39 - Diagrama de classes de dados para PTZ space options do serviço Media 135
Figura V.0.40 - Diagrama de classes de dados para PTZ control direction options do serviço Media 135
Figura V.0.41 - Diagrama de classes de dados para media profile do serviço Media 136
Figura V.0.42 - Diagrama de classes de dados para video source configuration do serviço Media . 136
Figura V.0.43 - Diagrama de classes de dados para video enconder configuration do serviço Media 136

Figura V.0.44 - Diagrama de classes de dados para MPEG4/H264 configuration information do serviço Media 137
Figura V.0.45 - Diagrama de classes de dados para multicast configuration do serviço Media....... 137
Figura V.0.46 - Diagrama de classes de dados para audio enconder configuration do serviço Media 138
Figura V.0.47 - Diagrama de classes de dados para metadata configuration do serviço Media 138
Figura V.0.48 - Diagrama de classes de dados para analytics engine configuration do serviço Media 139
Figura V.0.49 - Diagrama de classes de dados para PTZ configuration do serviço Media 139
Figura V.0.50 - Diagrama de classes de dados para video analytics configuration do serviço Media 140
Figura V.0.51 - Diagrama de classes de dados para rule engine configuration do serviço Media ... 140
Figura V.0.52 - Diagrama de classes de dados para capabilities do serviço PTZ 140
Figura V.0.53 - Diagrama de classes de dados para continuous move do serviço PTZ 141
Figura V.0.54 - Diagrama de classes de dados para absolute move do serviço PTZ 141
Figura V.0.55 - Diagrama de classes de dados para relative mode do serviço PTZ........................ 141
Figura V.0.56 - Diagrama de classes de dados para PTZ status information do serviço PTZ 142
Figura VI.0.1 - Diagrama de classes de serviço do Device Mamagment 143
Figura VI.0.2 - Diagrama de classes de serviço do Discovery 144
Figura VI.0.3 - Diagrama de classes de serviço do Events 144
Figura VI.0.4 - Diagrama de classes de serviço do Imaging 145
Figura VI.0.5 - Diagrama de classes de serviço do Media 146
Figura VI.0.6 - Diagrama de classes de serviço do PTZ.. 147
Figura VII.0.1 - Estrutura dos pedidos HTTP do tipo POST 149
Figura VII.0.2 - Formato de dados do HTTP Request.. 149
Figura VII.0.3 - Formato de dados do HTTP Response 150
Figura VII.0.4 - Conteúdo do envelope do pedido GetSystemDateAndTime 150
Figura X.0.1 - Dialog de adição de câmaras 160
Figura X.0.2 - Menu inicial de gestão de câmaras 160
Figura X.0.3 - Dialog Definitions com o Local Storage desabilitado.. 160
Figura X.0.4 - Dialog Definitions com o Local Storage habilitado 161

Figura X.0.5 - Tab users 161
Figura X.0.6 - Tab network........ 161
Figura X.0.7 - Tab system........ 161
Figura X.0.8 - Opções da Toolbar do serviço Device 162
Figura X.0.9 - Dialog das security capabilities........ 163
Figura X.0.10 – Dialog das system capabilities 163
Figura X.0.11 - Dialog das network capabilities 164
Figura X.0.12 - Dialog das miscellaneous capabilities........ 164
Figura X.0.13 - Capablities do serviço DeviceIO 164
Figura X.0.14 - Fragmento DeviceIO 165
Figura X.0.15 - Capabilities do serviço Media........ 165
Figura X.0.16 - Dialog de seleção de um profile 166
Figura X.0.17 - Tab Video........ 167
Figura X.0.18 - Tab Audio sem cartões suportados 167
Figura X.0.19 - Capabilities do serviço imaging........ 169
Figura X.0.20 - Dialog de seleção do video source 169
Figura X.0.21 - Tab settings do menu imaging........ 170
Figura X.0.22 – Dialog das PTZ capabilities........ 171
Figura X.0.23 - Dialog com mensagem de PTZ não suportado........ 171
Figura X.0.24 - Dialog com mensagem de serviço não implementado 172
Figura X.0.25 – Activity do menu Stream........ 172
Figura X.0.26 - Painel de navegação lateral 173

LISTA DE TABELAS

Tabela 2.1 - Caráter dos serviços NVT [4] 8

Tabela 2.2 - Pedidos realizados pela aplicação ONVIF IP Camera Monitor 19

Tabela 2.3 - Quantidade e tamanho de pedidos da aplicação ONVIF IP Camera Monitor 20

Tabela 2.4 - Pedidos realizados pela aplicação tinyCam Monitor FREE 22

Tabela 2.5 - Quantidade e tamanho de pedidos da aplicação tinyCam Monitor FREE 22

Tabela 2.6 - Pedidos realizados pela aplicação IP Cam Viewer Basic 24

Tabela 2.7 - Quantidade e tamanho de pedidos da aplicação IP Cam Viewer Basic 24

Tabela 3.1 - Configurações para negar todos os pedidos 33

Tabela 3.2 - Configurações para permitir todos os pedidos 33

Tabela 5.1 - Fragmentos de serviço simples 81

Tabela 5.2 - Fragmentos de serviço com fragmentos tabuladores 81

Tabela 5.3 - Views a usar por tipos de interações com os dados 82

Tabela 5.4 - Gestos e ícones por ação 83

Tabela 5.5 - Fragmentos de serviço com tabuladores 85

Tabela 5.6 - Métodos usados de acordo com as normalizações. 93

Tabela IX.0.1 - Tempos em milissegundos para o pedido GetMediaProfile 157

Tabela IX.0.2 - Tempos em milissegundos para o pedido SetSystemDateAndTime 157

Tabela X.0.1 - Dados do tabulador Vide de Media 167

Tabela X.0.2 - Dados do tabulador Audio de Media 168

Tabela X.0.3 - Dados do tabulador Metadata de Media 168

Tabela X.0.4 - Dados do tabulador Analytics de Media 168

Tabela X.0.5 - Dados do tabulador PTZ de Media 168

Tabela X.0.6 – Dados do tabulador ImageSettings de Imaging 170

LISTA DE ABREVIATURAS, SIGLAS E ACRÓNIMOS

API	Application Programming Interface
CGI	Common Gateway Interface
CRUD	Creade, Read, Update and Delete
GUI	Graphical User interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IO	Input Output
IP	Internet Protocol
IPC	Interprocess Communication
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, Tilt and Zoom
REST	Representational State Transfer
RGB	Red, Green and Blue
RTSP	Real Time Streaming Protocol
OO	Object Oriented
SFTP	SSH File Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer

TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	eXtensible Markup Language

1. INTRODUÇÃO.

1.1 Enquadramento

A existência e proliferação de inúmeros protocolos incompatíveis entre si de cada fabricante ou modelo, juntamente com o aumento da utilização de câmaras IP, fez surgir a necessidade de uniformizar a comunicação entre dispositivos IP de videovigilância. Foi por isso que surgiu o Open Network Video Interface (ONVIF) [3, 4]. O ONVIF proporciona normas para a comunicação entre dispositivos em rede. As especificações ONVIF estão disponíveis livremente de modo a promover a interoperabilidade de dispositivos independentemente dos fabricantes. Com a integração das normas ONVIF nos produtos, o mercado das câmaras IP sabe o que esperar do produto.

A tecnologia em que o ONVIF se baseia para comunicação são os serviços *web* SOAP que define modos normalizados de descrever protocolos de rede, formatos de transmissão de dados e operações a realizar.

Os dispositivos ONVIF implementam diferentes tipos de serviços, divididos em serviços obrigatórios, obrigatórios quando suportados e opcionais. Cada dispositivo tem associado a si um conjunto de serviços que o enquadram num tipo de dispositivo. Os dispositivos compatíveis com o ONVIF devem-se enquadrar em pelo menos um dos tipos de dispositivos. Os 4 tipos típicos de dispositivos ONVIF são NVD (Network Video Display), NVS (Network Video Storage), NVA (Network Video Analytics) e NVT (Network Video Transmitter).

Esta dissertação centra-se nos dispositivos NVT [5], ou seja, câmaras de vídeo ONVIF. Este tipo particular de dispositivos implementa obrigatoriamente os serviços Device Management, Events, Media e Device IO, o serviço PTZ quando têm essa funcionalidade, e opcionalmente os serviços Imaging e Video Analytics. Além destes, podem implementar serviços adicionais não NVT.

O Android é o sistema operativo mais utilizado nas plataformas móveis no mercado. Suporta diversos formatos de vídeo e áudio, resoluções de imagem, comunicação em rede. Além disso a capacidade de processamento e de armazenamento de dados nos dispositivos Android está a crescer. Os *tablets*, *phablets* e os *smartphones* conquistam cada vez mais o utilizador para desfrutar das funcionalidades multimédia do Android.

Em trabalhos anteriores [3,4], a comunicação com as câmaras é feita diretamente, o que significa que a aplicação Android implementa um cliente SOAP. O SOAP é pesado em consequência da serialização de dados XML. As aplicações envolvem uma arquitetura complexa que inclui uma biblioteca em C [5] de grande dimensão com cerca de 9Mb de código objeto, e uma camada JNI com diversas formas de trocar dados com as câmaras. As funcionalidades das duas aplicações desenvolvidas nos trabalhos supracitados são complementares, e a estrutura da interface de utilizador diverge.

Existe um serviço *web* ONVIF baseado na filosofia REST que é utilizado por uma aplicação web como intermediário de comunicação com as câmaras. O serviço é implementado sobre a biblioteca acima referida, que simplifica a interface das operações ONVIF e eleva o nível de abstração automatizando algumas operações, como por exemplo a gestão de perfis de Media. A API REST fornecida utiliza o formato de dados JSON e mapeia as funcionalidades ONVIF em recursos manipulados por verbos correspondentes aos 4 tipos principais de pedidos HTTP.

1.2 Motivação

As aplicações móveis atuais, e.g., [6]–[8], lidam com o peso computacional do ONVIF, o que reduz o desempenho computacional e a autonomia do dispositivo móvel [3], [4]. O ONVIF é baseado em normas de serviços web SOAP, o qual requer um poder computacional adicional. As câmaras IP comunicam através de dados XML, obrigando a uma serialização e de-serialização dos dados, tornando este processo crítico a nível de processamento. O formato JSON [9] é mais simples de processar e bastante mais compacto que o SOAP/ONVIF. A quantidade de informação que o cliente processa é assim reduzida através do servidor *web* REST existente. Testes realizados em ambiente controlado com implementações Android SDK demonstram uma redução no tempo e consumo de bateria comparativamente à alternativa SOAP/ONVIF [4]. Além disso, o servidor REST proporciona, por intermédio da biblioteca, uma abstração das operações ONVIF de gestão dos perfis de Media, que são as mais importantes e mais complexas. Assim, o servidor REST constitui uma oportunidade para reduzir também as mensagens enviadas e recebidas pela aplicação Android. A medição do ganho de desempenho desta funcionalidade não foi ainda medida, mas é certamente superior à obtida sem redução do número de mensagens e espera-se que seja de uma ordem de grandeza superior.

As funcionalidades complementares das duas aplicações existentes [3], [4] devem ser fundidas numa só, de modo a obter uma aplicação mais completa. A arquitetura dessa aplicação pode ser

melhorada, nomeadamente no que diz respeito ao armazenamento de dados locais. O modo de comunicação atual pode ser substituído para utilizar a aplicação servidora existente. Grande parte das funcionalidades aplicacionais para o serviço de Media encontram-se desenvolvidas assim como algumas funcionalidades para os restantes serviços, exceto, o Analytics e Events. Além disso é possível adicionar câmaras ONVIF e fazer as respetivas operações por câmara. Em alguns casos a UI das aplicações necessita de um reestruturação e atualização da mesma aproveitando algum do trabalho já desenvolvido.

1.3 Objetivos

O objetivo geral desta dissertação é o estudo, análise e desenvolvimento de uma solução para comunicação de aplicações Android e câmaras ONVIF utilizando um serviço web REST para o efeito. Além disso, essa solução deve procurar aumentar a eficiência computacional das operações. Pretende-se também integrar e implementar funcionalidades das normas ONVIF utilizando a mesma arquitetura de comunicação.

A arquitetura geral ajuda a relacionar vários componentes presentes neste trabalho com os objetivos delineados. Há componentes novos e existentes sendo que o desenho da arquitetura geral retrata a relação dos mesmos. Pretende-se adicionar e fazer melhorias de componentes presentes na arquitetura geral.

A arquitetura engloba o desenho da Biblioteca Java REST ONVIF e a restruturação de uma aplicação Android baseada em aplicações existentes desenvolvidas no seguimento de trabalhos anteriores. A biblioteca necessita de ser projetada sendo um componente inteiramente novo com base em estudos anteriores. A nova aplicação deve surgir do estudo de aplicações existentes no mercado e de trabalho anteriores.

Há 3 entidades na arquitetura idealizada que são a aplicação Android, o serviço web e o dispositivo NVT. A arquitetura é exibida na Figura 1.1 que retrata modos de comunicação entre entidades e descreve os componentes das mesmas.

Figura 1.1 - Arquitetura Geral do Sistema

A aplicação Android utilizará a Biblioteca Java REST ONVIF para a comunicação com o dispositivo NVT através do serviço web.

Os objetivos específicos são:

- Atualizar o servidor *web* REST existente para uma versão mais recente.
- Melhorar as configurações existentes do serviço *web* REST.
- Adicionar funcionalidades no serviço *web* REST.
- Desenvolver uma biblioteca Java que abstrai o serviço REST.
- O desenho da biblioteca deve prever o suporte para outras arquiteturas de comunicação a implementar futuramente.
- Desenvolver uma nova aplicação que integre as funcionalidades suportadas pelas 2 aplicações existentes e introduza melhoramentos nomeadamente na gestão de dados.
- Testar o consumo de memória em ambiente de rede real.

1.4 Estrutura da dissertação

O restante da presente dissertação é dividido em 5 capítulos.

No segundo capítulo referente ao Estado da Arte é destacado o trabalho relacionado com o tema desta dissertação. Descreve tecnologias como o ONVIF, servidor REST, API do servidor e as funções suportadas. É feita uma análise de aplicações existentes que utilizam serviços *web* SOAP e uma aplicação que utiliza alguns pedidos ao servidor REST. Por último é descrito as funções suportadas a arquitetura de *software* e da API REST do servidor *web* REST existente.

No terceiro capítulo referente ao servidor REST ONVIF é descrito o modo de funcionamento do servidor apache, processo de migração/instalação do servidor e destacada as funcionalidades adicionadas.

No quarto capítulo referente à Biblioteca Java REST ONVIF é descrito os requisitos, desenho da API e a implementação da mesma.

No quinto capítulo referente à aplicação Android é definido os requisitos e análise do desenho das aplicações existentes para integração na nova aplicação. Por último é descrito o desenho da nova aplicação com explicação das adições e melhorias realizadas.

No sexto e último capítulo referente às conclusões é feita uma descrição de resultados, conclusões e proposto trabalho futuro para dar continuação a este trabalho.

Para além dos capítulos existe uma seção de anexos tais como as novas configurações do servidor REST, documentação do servidor web REST, descrição da plataforma Android e documentação relativa à biblioteca Java REST ONVIF.

2. ESTADO DA ARTE

2.1 ONVIF

A existência de inúmeros protocolos, formatos e especificações utilizados por cada fabricante constituiu uma necessidade de um protocolo para normalizar a interação entre dispositivos. O *Open* Network Video Interface (ONVIF) [10] estabelece as normas para a comunicação de dispositivos em rede de modo a oferecer soluções para a fragmentação deste setor. As especificações ONVIF definem um protocolo comum para troca de informações entre dispositivos de captura de vídeo baseados em IP. O ONVIF surgiu em 2008 fundado pela Bosch Security Systems, Axys Communications e pela Sony Corporation [11].

O ONVIF é um protocolo utilizado em dispositivos IP que disponibiliza operações de diversos serviços, e.g., edição das configurações relativas a um perfil de media. Para obter soluções, as normas ONVIF têm uma plataforma subjacente que faz uso de serviços web SOAP [5].Este utiliza o formato estruturado de dados XML na troca de mensagens com o cliente. As normas focam-se na descoberta de dispositivo automático, configurações respetivas a diversos serviços (e.g., *streams* de vídeo), transferência de metadados com integridade que utiliza tecnologias como os serviços *web* e RTSP que é um protocolo de transferências de dados em tempo real. Estas normas suportam na íntegra serviços como análise de vídeo, controlo do áudio, reprodução de gravações e transmissão de áudio/vídeo em tempo real.

2.1.1 Tipos de dispositivos

Os dispositivos estão categorizados por tipos em que cada tipo de dispositivo implementa um conjunto de serviços. Os serviços podem ser obrigatórios, opcionais ou obrigatórios caso suportem características relacionadas. Assim, o tipo de serviço diz como é que o serviço se relaciona num determinado tipo de dispositivo.

Os dispositivos compatíveis com o ONVIF devem implementar pelo menos um dos tipos de dispositivos. Os 4 tipos principais de dispositivos ONVIF [1] são:

- **Network Video Display (NVD):** Dispositivo capaz de mostrar conteúdo recebido pela rede IP.

- **Network Video Storage (NVS):** Dispositivo capaz de armazenar conteúdo de uma câmara.
- **Network Video Analytics (NVA):** Dispositivo capaz de analisar o stream de vídeo e metadados recebidos de uma câmara IP.
- **Network Video Transmitter (NVT):** Dispositivo capaz de enviar conteúdos media através da rede IP, ou seja, uma câmara IP.

2.1.2 Serviços de dispositivos NVT

Os dispositivos NVT são os dispositivos de principal foco nesta dissertação deixando de lado os tipos NVS, NVA e NVD. Estes dispositivos podem implementar serviços ONVIF adicionais. A Tabela 2.1 [10] mostra o tipo de requisito para cada serviço de um NVT.

Tabela 2.1 - Caráter dos serviços NVT [2]

Tipo de serviço	Requisito
Device Managment	Obrigatório
Event	Obrigatório
Media	Obrigatório
Device IO	Obrigatório
Imaging	Opcional
PTZ	Obrigatório caso suportado
Video Analytics	Opcional

As normas ONVIF definem os vários serviços que compõem as funcionalidades de um NVT. O dispositivo pode suportar outros serviços e responder a pedidos de clientes através do serviço de descoberta do dispositivo.

2.1.3 Resumo dos Serviços

Os dispositivos NVT podem implementar uma gama de serviços que os define.

Device service fornece funcionalidades a um NVT que permitem gerir dispositivos, tais como as capacidades do dispositivo, configurações de rede e do sistema, configurações de segurança e atualizações de *firmware*.

Event service permite a um NVT enviar eventos para os clientes. Este serviço tem operações de subscrição e cancelamento de eventos. É ainda possível iniciar a transmissão/receção de eventos via Server Sent Events (SSE).

Media service permite a um NVT transmitir dados *media* para um cliente. Os dados *media* incluem som, vídeo, informação de análise de vídeo e outros metadados.

Device IO service como o próprio nome sugere, permite a um NVT suportar entradas e saídas físicas do dispositivo.

PTZ *service* permite a um NVT providenciar controlo PTZ no caso de um dispositivo implementar PTZ. O PTZ oferece às câmaras as funcionalidades de *pan*, *tilt* e *zoom*.

Imaging service permite a um NVT a configuração das definições de imagem que afetam o vídeo em termos visuais, por exemplo, tempo de exposição, ganho e *white balance* (processo de remoção de cores não reais) e controlo de foco.

Video Analytics service permite a um NVT fornecer funcionalidades de análise de vídeo. Um exemplo é a obtenção de metadados com informação sobre a deteção de objetos como o tipo de objeto, hora, centro de gravidade e coordenadas de delimitação.

Um exemplo de serviços adicionais que podem ser implementados pelos dispositivos NVT é o serviço de gravação de vídeo no caso de existir suporte para armazenamento de dados.

2.2 Serviços REST para ONVIF

Numa pesquisa por servidores REST ONVIF não se conseguiu encontrar serviços *web* REST para comunicação com câmaras IP ONVIF. Devido a este facto, neste capitulo, é feita uma análise de um mapeamento REST. Além disso é feita uma descrição de bibliotecas Clientes ONVIF de modo a perceber o seu desenho e modo de funcionamento. A implementação do serviço web REST corresponde a trabalho relacionado, pelo que, é feita a descrição da arquitetura e da API.

2.2.1 Serviços *web* REST

Analisaram-se mapeamentos REST em geral de modo a fazer um estudo complementar ao do servidor *web* existente. A análise incide sobre como se define entidades e como são utilizados os verbos.

Os princípios de Representational State Transfer (REST) [12][13] utilizam entidades e funcionalidades conceptuais modeladas em URIs (Universal Resource Indentifiers) que são recursos

identificadores. Estes recursos podem ser acedidos pelas operações HTTP (p.e GET, também conhecida por verbo), sendo que, geralmente são usadas as conhecidas funcionalidades CRUD (*create*, *read*, *update* e *delete*) para operar. Às operações de obter, criar, editar e eliminar associam-se respetivamente aos verbos GET, POST, PUT e DELETE. Os componentes de um sistema REST comunicam através destas operações com uso de recursos, em que, as aplicações clientes e servidoras passam por diversos estados de recursos de acordo com as ligações entre eles, baseados na sequência de pedidos.

Numa análise feita a um mapeamento REST [14] verificou-se que se fez uma divisão de recursos em 3 tipos de acordo com o tipo de serviço que providenciam. O primeiro tipo é "Resourse Set Service" que diz que o serviço é mapeado para um conjunto de recursos de domínio, por exemplo, um recurso "CustumerSet". Este tipo de recurso pode suportar as quatro operações HTTP (GET, PUT, POST e DELETE). O segundo tipo, "Individual Resource Service", é para recursos de domínio individual no conjunto de recursos, por exemplo, o recurso "Custumer" é mapeado para este tipo. Este tipo de recurso pode suportar as operações GET, PUT e DELETE sendo que o POST não é aplicável, uma vez que, o recurso individual URI identificado já está criado. O terceiro tipo, "Transitional Service", é para serviços que consomem recursos, criam recursos e atualizam/transformam os estados dos recursos relacionados, por exemplo, o recurso "SubmitPayment". Quando se invoca o "SubmitPayment" com as informações do pedido cria-se um novo "*payment*" associado a uma compra onde se atualiza o estado de pagamento "*isPaid*" para verdadeiro. O terceiro tipo só suporta a operação POST e pode ser utilizado para capturar funcionalidades orientadas à transição. O exemplo completo do mapeamento do serviço *web* REST analisado pode ser consultado na Figura 2.1 onde é possível identificar o tipo de recurso, URI definida e as operações suportadas.

RESTful WSs		
CustomerSet	TYPE:	Type I
	URI:	http://some.com/Customers
	Supported Operations:	GET, PUT, DELETE, POST
OrderSet	TYPE:	Type I
	URI:	http://some.com/Customers/[Customer_id]/orders
	Supported Operations:	GET, PUT, DELETE, POST
Customer	TYPE:	Type II
	URI:	http://some.com/Customers/[Customer_id]
	Supported Operations:	GET, PUT, DELETE
Order	TYPE:	Type II
	URI:	http://some.com/customers/[customer_id]/orders/[order_id]
	Supported Operations:	GET, PUT, DELETE
Payment	TYPE:	Type II
	URI:	http://some.com/customers/[customer_id]/orders/[order_id]/payment
	Supported Operations:	GET, PUT, DELETE
Shipment	TYPE:	Type III
	URI:	http://some.com/customers/[customer_id]/orders/[order_id]/shipment
	Supported Operations:	POST
SubmitPayment	TYPE:	Type III
	URI:	http://some.com/customers/[customer_id]/orders/[order_id]/submitpayment
	Supported Operations:	POST
ShipOrder	TYPE:	Type III
	URI:	http://some.com/customers/[customer_id]/orders/[order_id]/shipOrder
	Supported Operations:	POST

Figura 2.1 - Mapeamento REST exemplo [5].

2.2.2 Bibliotecas Cliente ONVIF

Não se encontraram bibliotecas cliente REST ONVIF, o que é expectável, pois se não há servidores, não há clientes nem bibliotecas. O que existe são bibliotecas ONVIF para comunicação direta com as câmaras para diferentes plataformas e nenhuma especifica para Android através de servidor REST, e que são resumidamente apresentadas na secção 2.2.2 Bibliotecas Cliente ONVIF.

Na pesquisa por bibliotecas cliente ONIVF encontraram-se 5 bibliotecas principais nomeadamente a "Happytime Onvif Client", "onvif-java-lib", "python-onvif", "onvifcpplib" e "Client to ONVIF NVT devices Profile S: cameras". as quais podem ser usadas na plataforma Android. Estas utilizam o serviço *web* SOAP para comunicação direta com os dispositivos ONVIF. Analisaram-se bibliotecas cliente ONVIF que são usadas no processo de comunicação entre as aplicações e NVT. A biblioteca "Client to ONVIF NVT devices Profile S: cameras" tem a particularidade de ser para a plataforma servidora Node.JS. A popular aplicação "Onvif Device Manager" [15] para computadores de secretária não separa/contém uma biblioteca ONVIF. Seguidamente, é feito uma análise do tipo das linguagens utilizadas, modo de comunicação, operações suportadas e nível de abstração das bibliotecas.

Happytime Onvif Client

A biblioteca "Happytime Onvif Client" [16] foi desenvolvida em C, suporta diferentes plataformas, entre elas o Android NDK, e pode ser facilmente embebida noutros sistemas. Além disso, não recorre a bibliotecas de terceiros. Esta disponibiliza suporte para as especificações ONVIF NVT: implementa descoberta de dispositivos, os serviços Device Managment, Media, Imaging, Events, Analytics e PTZ. A biblioteca tem suporte para descodificar os formatos de vídeo H265/H264/MJPEG/MPEG4 e de áudio G711/G726/ACC. Para utilizar a biblioteca e o código fonte é necessário efetuar a compra dos mesmos.

onvif-java-lib

A "onvif-java-lib" [17] é uma biblioteca desenvolvida na linguagem de programação Java. A biblioteca encontra-se em desenvolvimento sendo que existem funcionalidades por implementar. Existem classes para os serviços Imaging, Media, PTZ e Device Managment. As classes de serviço são instanciadas pela classe OnvifDevice que necessita das informações da câmara como as credenciais e o endereço *web.* As classes para os serviços não estão completas com suporte para todos os pedidos possíveis de um serviço. Adicionalmente existem classes para o serviço Discovery.

python-onvif

A "python-onvif" [18] é uma biblioteca desenvolvida na linguagem de programação Python. Esta disponibiliza classes para criar objetos relativos à câmara e aos serviços. Os objetos de serviço disponibilizam métodos que abstraem a comunicação com a câmara. Os métodos recebem/devolvem objetos de classes usadas para representar a informação a enviar/receber nos pedidos. A instanciação do objeto "ONVIFCamera" implica, por defeito, a inicialização do objeto de serviço referente ao Device Managment que disponibiliza todas as operações respetivas ao mesmo. A biblioteca disponibiliza um modo de instanciar objetos de serviço ONVIF e outro modo para serviços não oficiais. Em adição, com a instalação dos módulos do repositório da biblioteca, é possível testar as operações da mesma através da linha de comandos com os modos interativo e em *batch.*

onvifcpplib

A "onvifcpplib" [19] é uma biblioteca baseada em gSOAP [20], que é uma ferramenta de desenvolvimento de *software* em C e C++ para serviços *web* SOAP e REST e ligações de dados entre XML e C/C++. Esta biblioteca foi gerada a partir dos *schema* e WSDL ONVIF, e tem suporte para ProfileS, ProfileG e Events. A câmara é abstraída pela classe OnvifClientDevice que deve ser

instanciada para efetuar a comunicação com a mesma. Podem-se usar assim classes para os serviços de um NVT, métodos dos serviços e classes destinadas à gestão de dados.

Client to ONVIF NVT devices Profile S: cameras [21]

Esta biblioteca é para desenvolver clientes ONVIF e atua no lado do servidor, a qual, interpreta JavaScript em tempo de execução através do Node-JS. A biblioteca disponibiliza funcionalidades para obter informações de dispositivos NVT, assim como, *streams* de media, controlar os movimentos PTZ, gerir *presets,* descobrir dispositivos numa rede e receber eventos. Também tem suporte para dispositivos NVR Profile G sendo possível obter a lista de gravações. Para todos os serviços da biblioteca, respetivamente o Device Managment, Imaging, Media e PTZ, há pedidos não implementados. Na API existem apenas as classes "Cam" e "Discovery". A classe "Cam" disponibiliza os métodos dos serviços dos dispositivos. A classe "Discovery" disponibiliza um método de *probe* com atributos de entrada como o *timeout, resolve* e *message* ID. Existe uma *callback* relacionada ao método que permite lidar com as câmaras descobertas na rede através de *probes.* Apesar de existirem falta de funcionalidades a biblioteca está bem documentada.

2.2.3 Arquitetura do software

O serviço *web* fornece um modo de comunicação entre aplicações clientes e NVT que diminui a carga computacional do cliente. O protocolo de comunicação usado é o HTTP com suporte para comunicação cifrada (HTTPS), e o JSON é usado para o envio de informação estruturada. A capacidade de abstração do serviço *web* proporciona um modo de comunicação entre aplicações independente da plataforma e linguagem de programação do cliente. A arquitetura REST é ideal para aplicações hipermédia em rede, sendo usada na construção de serviços web que são leves, de fácil manutenção e escaláveis. Nesta secção é feita a descrição da arquitetura do *software* do servidor, arquitetura da API REST e uma descrição das funções suportadas pela API.

Em trabalhos anteriores [4] demonstrou-se que a arquitetura REST tem benefícios no consumo de carga e eficiência no processamento. Esta reduz a quantidade de informação dos pedidos utilizando o formato de dados estruturado JSON.

A aplicação servidora que implementa o serviço REST está desenvolvida na linguagem C++, na forma de um FastCGI e da biblioteca C UMOC [5]. A biblioteca UMOC implementa a troca de mensagens SOAP com a câmara. A aplicação está alojada num servidor HTTP Apache com um módulo SSL para comunicação cifrada. A arquitetura do servidor REST ONVIF é exibida na Figura 2.2.

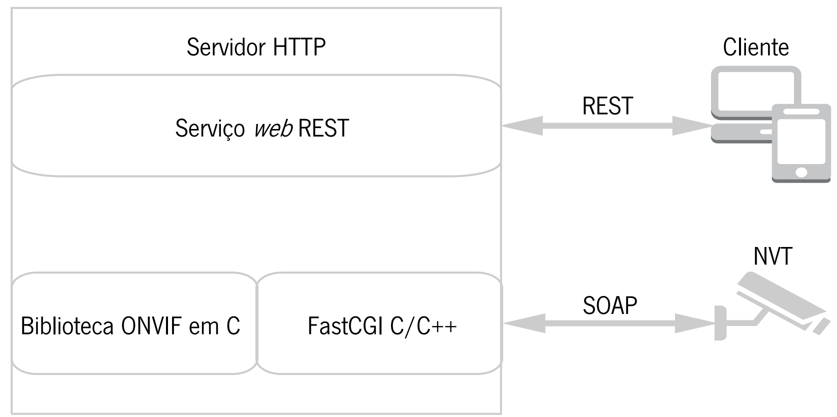


Figura 2.2 - Arquitetura do servidor REST ONVIF

O serviço *web* disponibiliza as funcionalidades da biblioteca UMOC numa *interface* REST, fornecendo uma API *web* mais simples que os serviços ONVIF (SOAP/XML). Este trata da comunicação com câmaras IP, nomeadamente da serialização e desserialização dos dados XML, análise e validação das mensagens SOAP e fornece funcionalidades que têm um nível de abstração superior aos comandos ONVIF. Assim, a aplicação cliente tem uma carga computacional menor ao utilizar o serviço *web* REST. Na comunicação com o cliente, o serviço serializa dados no formato JSON, reduzindo significativamente o volume de tráfego e o processamento necessário.

O NVT é o dispositivo ONVIF que comunica com o servidor através do serviço *web* SOAP. A particularidade deste dispositivo é a utilização do protocolo ONVIF e de ser um NVT podendo variar na quantidade de serviços implementados.

A comunicação através do servidor REST é sempre iniciada por um pedido HTTP do cliente, usando o método e URI pretendidos. No cabeçalho são enviadas as credenciais de acesso às câmaras com os parâmetros “usr” e “pwd” correspondente ao nome de utilizador e palavra-chave. O servidor devolve um HTTP Response com conteúdo JSON. Quando o servidor processa o pedido do cliente envia e recebe mensagens SOAP à câmara, utilizando como protocolo de transporte o HTTP e como formato de dados o XML. As entidades e a sequência dos pedidos é exibida na Figura 2.3.

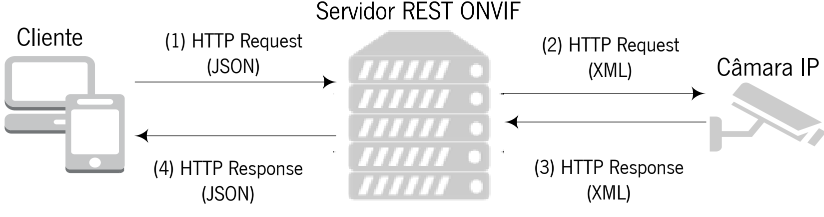


Figura 2.3 - Diagrama temporal de um pedido ao servidor

O exemplo do diagrama temporal da Figura 2.3. representa o tipo de pedido mais simples a nível de mensagens SOAP porque alguns pedidos podem envolver várias mensagens entre o servidor e o dispositivo ONVIF. O cliente tem apenas duas mensagens, correspondentes ao pedido realizado e à resposta do servidor, sendo que ambas, conjugam todas as mensagens trocadas entre NVT e cliente numa só mensagem.

2.2.4 Arquitetura do serviço REST existente

A API REST do servidor está estruturada por serviços com várias operações. A API implementada utiliza 4 métodos HTTP, que são:

- **GET**: usado para obter informações de um recurso.
- **POST**: usado para criar recursos com determinadas informações.
- **PUT**: usado para editar informações de recursos. Em alguns casos cria recursos.
- **DELETE**: usado para eliminar recursos.

O acesso a um recurso ONVIF é realizado através de uma URI no formato "/onvif/{serviço_onvif},{endereço_serviço_codficado}" em que:

- **{serviço_onvif}:** Um dos serviços ONVIF podendo ser o devicemgmt, media, imaging, ptz, deviceio e events.
- **{endereço_serviço_codficado}:** Endereço completo da câmara sem especificar o protocolo (i.e., sem "http://") e codificado (*url encoded*).

Por exemplo, para o método GET da operação de obter *scopes* do serviço Device Managment, tem-se a URI "/onvif/devicemgmt,193.136.12.31%2Fonvif%2Fdevice_service/scopes.".

O URI base do serviço Device Managment é fundamental, uma vez que, é através dele que se pode obter todos os outros URIs através do método GET. Assim, o cliente só precisa conhecer a URI base para aceder aos restantes URIs.

As normas ONVIF estão em constante desenvolvimento fornecendo novas funcionalidades. A aplicação servidora existente implementa diversas funcionalidades para os serviços Device Managment, Device IO, Imaging, Media e PTZ. Os serviços Discovery, Events e Analytics encontram-se em desenvolvimento pelo que ainda não integram o servidor. A descrição das funções suportadas e por implementar é feita no Anexo III – Funções suportadas pelo Serviço web REST. É feita uma análise das funções suportadas comparativamente às das especificações ONVIF mais atuais [22]. Existem

funcionalidades por implementar para todos os serviços de um NVT. As especificações ONVIF disponibilizam um índice de operações por serviço que permite consultar a documentação das operações.

2.3 Aplicações Android para ONVIF

Existem inúmeras aplicações que fazem uso de diversas funcionalidades ONVIF. O critério de escolha das aplicações para análise foi essencialmente suporte ONVIF, número de transferências e a cotação atribuída pelos utilizadores da loja de aplicações, uma vez que, estas são geralmente mais bem concebidas. Pretende-se testar funcionalidades como obter as capacidades ou serviços de um dispositivo. As aplicações analisadas presentes no mercado comunicam diretamente com as câmaras o que significa que usam comunicação SOAP. Os processos de serialização e desserialização de dados XML reduz significativamente a performance das aplicações. Nesta secção pretende-se fazer uma análise de outros factos de modo a entender melhor o ponto de situação das aplicações atuais.

As aplicações atuais comunicam com as câmaras IP diretamente sem a utilização de um servidor intermédio. A análise de funcionalidades destas aplicações foi feita em trabalhos relacionados. Neste capítulo é feita a análise de tráfego para estudar o comportamento das mesmas. As aplicações podem ser obtidas essencialmente a partir da loja de aplicações (*Google Play*) da Google.

Utilizou-se o Wireshark para analisar o tráfego das aplicações, de modo a detetar se comunicavam SOAP através da análise do conteúdo dos pedidos. Antes de analisar as aplicações executou-se o programa tPacketCapture [23] para Android de forma a capturar o tráfego das aplicações. A aplicação gerou um ficheiro no formato "pcap" com a informação do tráfego gerado durante a captura. Posteriormente analisou-se o ficheiro obtido com auxilio das ferramentas do Wireshark. Através da análise de tráfego verificou-se que todas as aplicações utilizam a comunicação ONVIF.

Quando se iniciou a captura de tráfego das aplicações configurou-se uma câmara e testou-se as diferentes funcionalidades uma vez. Verificou-se que grande parte dos pedidos eram realizados quando se configurava a câmara. Para repetir esses pedidos era necessário reconfigurar outra câmara. Sendo assim procedeu-se do mesmo modo na geração de tráfego para todas as aplicações testando uma só vez as funcionalidades disponíveis e recolhendo os pacotes gerados.

2.3.1 ONVIF IP Camera Monitor

A aplicação ONVIF IP Camera Monitor [6] é uma aplicação que permite monitorizar, explorar e configurar câmaras IP. À data, esta aplicação tem mais de 50 mil transferências e uma cotação de 4,3 avaliada por 532 pessoas. Os serviços NVT suportados são o Device Managment, Media, Imaging, PTZ e Discovery. É possível visualizar diversos vídeos de várias câmaras em simultâneo (máximo de 6:6), comunicação SSL/HTTPS e tem suporte para visualização *landscape* e *portrait.* Para além da aplicação principal é disponibilizado um Widget para visualização da stream continuamente no Android.

A tela inicial da aplicação apresenta os dispositivos adicionados (Figura 2.5), contém botões para adicionar dispositivos, aceder ao vídeo das câmaras, configurar quantidade de *streams* em simultâneo para visualização num só ecrã e para pesquisar informações sobre diferentes tipos de dispositivos ONVIF. A Figura 2.4 mostra um exemplo de adição de uma câmara IP à aplicação. Depois disso, a aplicação efetua diversos pedidos, como o getCapabilities e getServices.

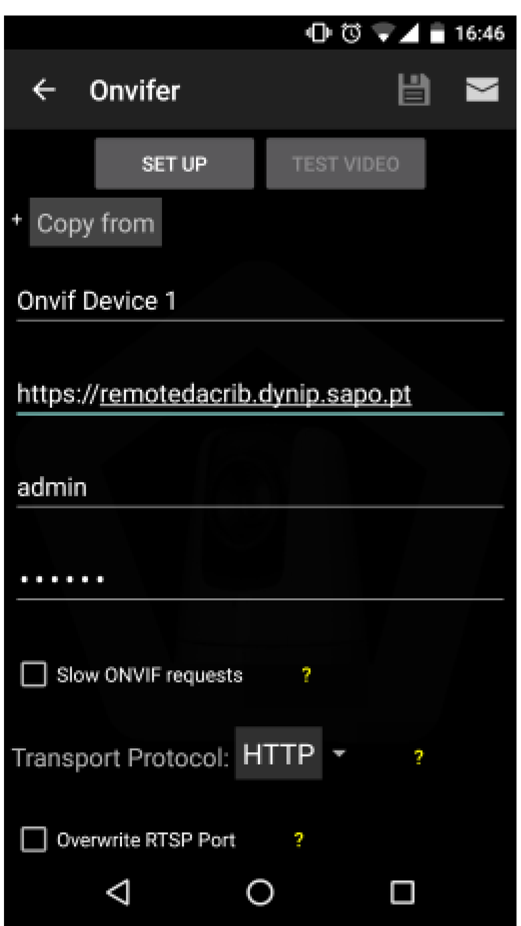


Figura 2.4 - Menu para adicionar uma câmara

Figura 2.5 - Menu inicial com as câmaras disponíveis

Com auxilio do Wireshark aplicaram-se filtros para o conteúdo de cabeçalho e de corpo HTTP do tráfego gerado pela aplicação. A Figura 2.6 ilustra o conteúdo do HTTP Header do pedido GetSystemDateAndTime.

```
<v:Header>
  <Action
    mustUnderstand="1"
    xmlns="http://www.w3.org/2005/08/addressing">
    http://www.onvif.org/ver10/device/wsdl/GetSystemDateAndTime
    </Action>
  </v:Header>
```

Figura 2.6 - Conteúdo do HTTP *header*

Obteve-se as informações da camada HTTP para o tráfego de um pedido representado na Figura 2.7 que ilustra um método POST com URI "/onvif/device_service" que corresponde ao serviço Device Managment. O agente de *software* utilizado foi o ksoap2-android para lidar com WSDL e SOAP. É ainda possível ver informações acerca do *enconding*, tipo de conteúdo, conexão, tamanho e endereço.

```
Hypertext Transfer Protocol
  POST /onvif/device_service HTTP/1.1\r\n
  User-Agent: ksoap2-android/2.6.0+\r\n
  Content-Type: application/soap+xml;charset=utf-8\r\n
  Accept-Encoding: gzip\r\n
  Connection: close\r\n
  Content-Length: 471\r\n
  Host: 193.136.12.31\r\n
  \r\n
  [Full request URI: http://193.136.12.31/onvif/device_service]
  [HTTP request 1/1]
  [Response in frame: 609]
```

Figura 2.7 - Dados HTTP do pedido GetSystemDateAndTime

O tamanho total em bytes é obtido pela soma dos dados enviados e recebidos. A Figura 2.8 tem o tamanho total, enviado e recebido do pedido GetSystemDateAndTime.

```
Entire conversation (3521 bytes)
Entire conversation (3521 bytes)
193.136.12.31:80 → 10.8.0.1:43567 (2842 bytes)
10.8.0.1:43567 → 193.136.12.31:80 (679 bytes)
```

Figura 2.8 - Tamanho em bytes do pedido GetSystemDateAndTime

Testou-se as diversas funcionalidades ONVIF da aplicação de modo a obter o tráfego relevante. Através da análise dos pacotes com as funcionalidades do Wireshark obteve-se os dados da Tabela 2.2

e Tabela 2.3. A Tabela 2.3 mostra que foram realizados diversos tipos de operações ONVIF (1ª coluna), e que usam determinados caminhos da URI (4ª coluna).

Os pedidos do tipo GET não têm conteúdo de corpo no pedido realizado à câmara e não são SOAP/ONVIF. O tipo de pedido é identificado com base na URI.

Tabela 2.2 - Pedidos realizados pela aplicação ONVIF IP Camera Monitor

Action	Caminho da URI
GetSystemDateAndTime	POST /onvif/device_service HTTP/1.1
GetDeviceInformation	
GetCapabilities	
GetServices	
GetProfiles	POST /onvif/Media HTTP/1.1
GetVideoSources	
GetVideoSourcesInformation	
GetVideoSourcesConfigurations	
GetVideoEncodeConfigurations	
GetAudioSources	
GetAudioSourcesConfigurations	
GetAudioEncoderConfigurations	
DeleteProfile	
CreateProfile	
AddVideoSourceConfiguration	
GetProfile	
AddVideoEncoderConfiguration	
GetProfiles	
GetSnapshotUri	
GetStreamUri	
	GET /Streaming/Channels/101?transportmode=unicast&profile=Profile_1 HTTP/1.0
	GET /onvif-http/snapshot?Profile_1 HTTP/1.1

A Tabela 2.3 tem as informações sobre a quantidade e respetivos tamanhos sobre os diversos pedidos feitos pela aplicação. Alguns pedidos foram realizados mais que uma vez como o GetSystemDateAndTime que tem 5 registos de pedidos. O tamanho total de dados recebidos e enviados é significativo relativamente ao tamanho de dados utilizados pela aplicação. Não se registou o

tamanho de alguns pedidos do tipo GET porque as respostas são dados contínuos, p.e, uma *stream* de vídeo.

Tabela 2.3 - Quantidade e tamanho de pedidos da aplicação ONVIF IP Camera Monitor

Action	**Quantidade**	**Tamanho [httpreq/httpres] (bytes)**	**Tamanho total (bytes)**
GetSystemDateAndTime	5	679/2842	3521
GetDeviceInformation	1	1170/2415	3585
GetCapabilities	1	1200/6566	7766
GetServices	1	1207/6736	7943
GetProfiles	2	1141/13k	14k
GetVideoSources	1	1149/3283	4432
GetVideoSourcesConfigurations	1	1175/2498	3673
GetVideoEncodeConfigurations	1	1177/3771	4948
GetAudioSources	1	1149/2428	3577
GetAudioSourcesConfigurations	1	1175/2428	3603
GetAudioEncoderConfigurations	1	1177/2428	3605
DeleteProfile	2	1200/2114	3314
CreateProfile	2	1210/2267	3477
AddVideoSourceConfiguration	2	1299/2128	3427
GetProfile	1	1190/2606	3796
AddVideoEncoderConfiguration	2	1305/2129	3434
GetSnapshotUri	2	1201/2384	3585
GetStreamUri	3	1402/2421	3823

Pedidos como o GetProfiles e GetCapabilities, que apresentam maior volume de informações, podem ter efeitos significativos a nível de desempenho. A quantidade repetida de pedidos contribui para o aumento do volume de dados. Pedidos mais básicos da aplicação apresentam um volume de informação menor. Verifica-se a existência de pedidos repetidos resultando em redundância na comunicação.

2.3.2 tinyCam Monitor FREE

A tinyCam Monitor FREE [8] é uma aplicação que permite explorar funcionalidades das câmaras IP. À data, esta aplicação tem mais de 5 milhões de transferências e uma cotação de 4,0 avaliada por quase 40 mil pessoas. Suporta funcionalidades PTZ, agrupamento da *stream* de vídeo e

comunicação com SLL/HTTPS. Para além de câmaras ONVIF também é possível configurar câmaras de fabricantes com protocolos diferentes.

A aplicação tem um menu principal exibido na Figura 2.10 que efetua de imediato a *stream* de vídeo das câmaras adicionadas. Existem opções adicionais para explorar funcionalidades como áudio ou PTZ. O menu principal tem uma barra de navegação lateral identificada na Figura 2.9 onde se acede às opções para gerir câmaras e definições gerais da aplicação.

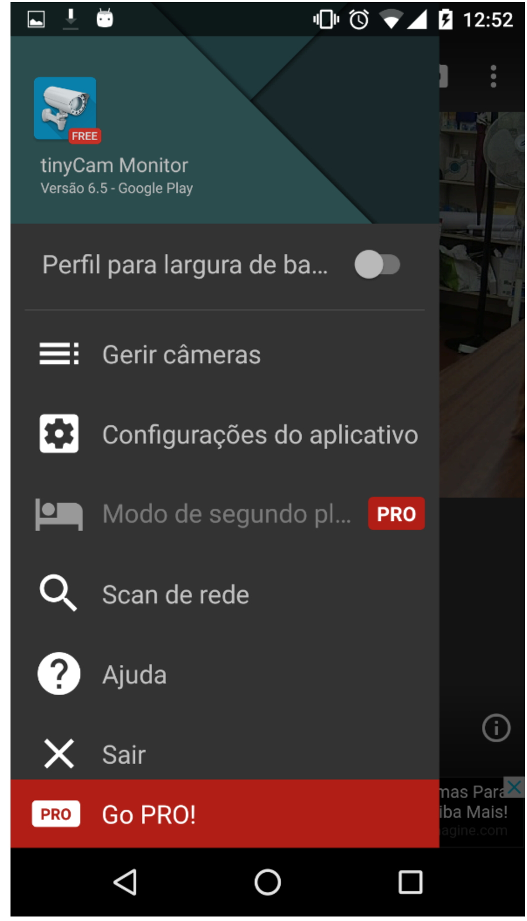


Figura 2.10 - Navigation Drawer do menu inicial

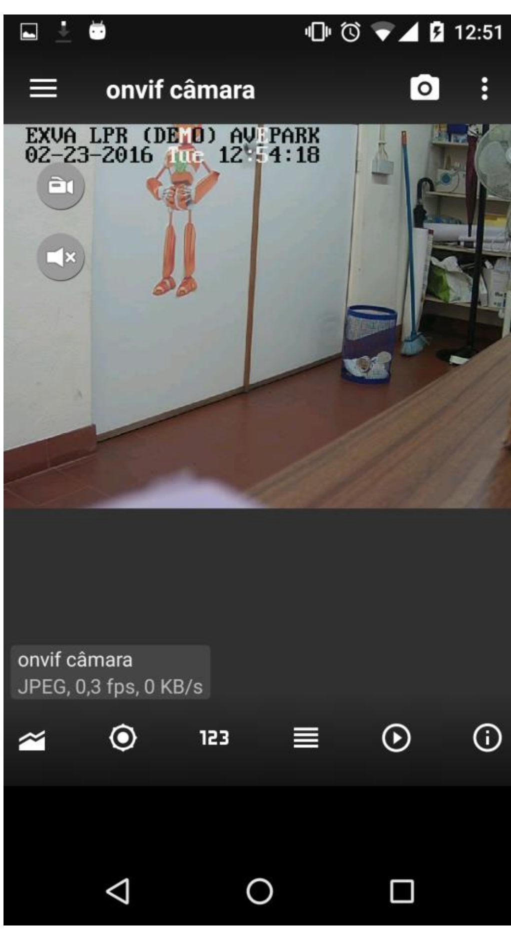


Figura 2.9 - Menu inicial com câmara configurada

A aplicação utiliza SOAP para a comunicação. O *user-agent* das mensagens HTTP é o Android/6.0.1, o que parece indicar que não foi utilizada qualquer biblioteca SOAP, mas sim e apenas a biblioteca HTTP nativa do Android.

Tabela 2.4 - Pedidos realizados pela aplicação tinyCam Monitor FREE

Action	**Caminho da URI**
GetCapabilities	POST /onvif/device_service HTTP/1.1
GetProfiles	POST /onvif/Media HTTP/1.1
GetStreamUri	
GetSnapshotUri	
GetOptions	POST /onvif/Imaging HTTP1.1
GetImagingSettings	
SetImagingSettings	
	GET /videostream.cgi HTTP/1.1
	GET /decoder_control.cgi?command=[code] HTTP/1.1
	GET /Streaming/Channels/101?transportmode=unicast&profile=Profile_1 HTTP/1.1
	GET /onvif-http/snapshot?Profile_1 HTTP/1.1

A aplicação repetiu uma vez todos os pedidos para obter dados como as *capabilities* ou *profiles*. A quantidade e tamanho de dados estão registados na Tabela 2.5.

Tabela 2.5 - Quantidade e tamanho de pedidos da aplicação tinyCam Monitor FREE

Action	**Quantidade**	**Tamanho [httpreq/httpres] (bytes)**	**Tamanho total (bytes)**
GetCapabilities	2	1243/6566	7809
GetProfiles	2	1183/18k	19k
GetStreamUri	2	1425/2421	3846
GetSnapshotUri	2	1243/2384	3627
GetOptions	2	1249/3315	4564
GetImagingSettings	2	1273/2796	4069
SetImagingSettings	2	1338/2120	3458

2.3.3 IP Cam Viewer Basic

A aplicação IP Cam Viewer Basic [7] é uma aplicação que permite monitorizar, explorar e configurar câmaras IP. À data, esta aplicação tem mais de 1 000 000 transferências e uma cotação de 4,1 avaliada por 14742 pessoas. Suporta funcionalidades PTZ, todos os formatos de vídeo ONVIF (MPEG4, H264, MJPEG), agrupamento da *stream* e comunicação com SLL/HTTPS. A aplicação tem funcionalidades para a versão *pro* que não influenciam de forma crítica a análise da mesma. Para além de câmaras ONVIF também é possível configurar câmaras com suporte para protocolos diferentes.

O menu principal representado na Figura 2.11 é exibido quando se inicia a aplicação. Pode-se visualizar a *stream* e aceder às diversas funcionalidades das câmaras. Ao adicionar uma câmara para além do IP e tipo de modelo da câmara é necessário especificar a porta (mesmo que sejam portas por defeito 80/443). Para as aplicações "ONVIF IP Camera Monitor" e "tinyCam Viewer Basic" não é necessário especificar a porta.

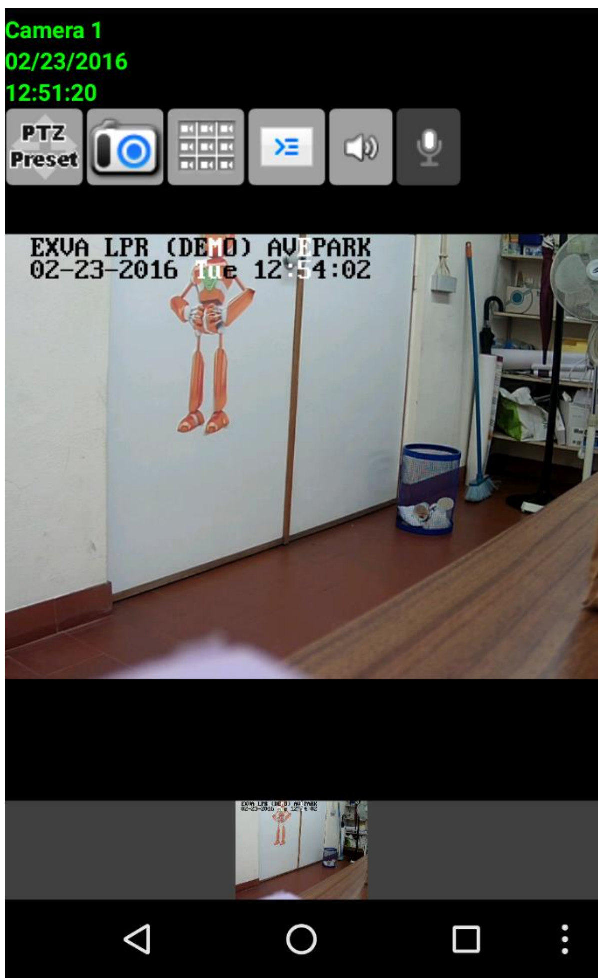


Figura 2.11 - Menu principal com uma câmara configurada

Verificou-se que a aplicação utiliza SOAP na comunicação. O formato do conteúdo HTTP corresponde ao das especificações ONVIF.

Tabela 2.6 - Pedidos realizados pela aplicação IP Cam Viewer Basic

Action	Caminho da URI
GetDeviceInformation	POST /onvif/device_service HTTP/1.1
GetCapabilities	
GetReayOutputs	
GetProfiles	POST /onvif/Media HTTP/1.1
GetSnapshotUri	
GetStreamUri	
	GET /Streaming/Channels/101?transportmode=unicast&profile=Profile_1 HTTP/1.0
	GET /onvif-http/snapshot?Profile_1 HTTP/1.1

A aplicação repete menos pedidos. A quantidade e tamanho de dados podem ser consultados na Tabela 2.7.

Tabela 2.7 - Quantidade e tamanho de pedidos da aplicação IP Cam Viewer Basic

Action	Quantidade	Tamanho [httpreq/httpres] (bytes)	Tamanho total (bytes)
GetDeviceInformation	1	1301/2415	3716
GetCapabilities	1	1337/6566	7903
GetRelayOutputs	1	1296/2145	3441
GetProfiles	1	1273/14k	15k
GetSnapshotUri	1	1330/2384	3714
GetStreamUri	3	1514/2421	3935

2.3.4 Sumário

As aplicações analisadas apresentam diferenças a nível de funcionalidades, tecnologias utilizadas e performance. Nenhuma aplicação é dedicada exclusivamente para dispositivos ONVIF e existem aplicações com suporte para câmaras que utilizam protocolos desenvolvidos pelo fabricante. A aplicação ONVIF IP Camera Monitor é a disponibiliza maior quantidades de operações. No entanto, transfere mais dados que outras aplicações, que realizam um menor número de pedidos. As aplicações não suportam todos os serviços disponíveis por uma câmara. A nível de defeitos todas a aplicações apresentam pedidos com e sem redundância. Os pedidos repetidos influenciam o

desempenho ao executar tarefas de rede extra que corresponde a uma maior quantidade de dados. A aplicação "ONVIF IP Camera Monitor" faz uso de uma biblioteca ksop2-android [24] que é leve e estável. O tamanho dos pedidos permite ainda identificar diferenças significativas nos tamanhos em relação a pedidos da arquitetura REST. Verificou-se que os métodos getProfiles e getCapabilities são os que apresentam maior volume de dados para todas as aplicações.

2.3.5 Aplicações base

Existem aplicações relacionadas [3], [4] que surgem de trabalhos anteriores. Pretende-se continuar o trabalho realizado pelo que é feita uma análise das tecnologias e funcionalidades das aplicações desenvolvidas anteriormente.

Aplicação com suporte para alguns serviços

A aplicação descrita em [3] utiliza o NDK e comunicação direta por JNI utilizando a biblioteca UMOC. Por isso, utiliza mensagens SOAP. Está desenhada para proporcionar acesso a informações da câmara e para alterar configurações dos serviços Device Managment, Device IO, Media e Imaging. O menu principal da aplicação proporciona um modo de adicionar câmaras e de as explorar. Quando se seleciona uma câmara é possível aceder às funcionalidades dos serviços através de um painel de navegação lateral com ligações a cada um dos serviços. É possível explorar diversas funcionalidades de cada serviço como adicionar *scopes,* ver informações de *capabilites* ou visualizar *stream* de vídeo.

A nível de arquitetura da aplicação, não existe uma camada de comunicação orientada a objetos, mas sim uma coleção de funções *static* nativas/JNI sem relação entre si. As funções comunicam com as câmaras para enviar e receber informações. Existe ainda uma estrutura de dados para guardar as informações obtidas.

Aplicação para o serviço Media

A aplicação para o serviço media [4] usa a mesma API JNI para comunicação com as câmaras. A principal funcionalidade é explorar as funções do serviço de Media de um dispositivo ONVIF. O menu inicial da aplicação permite a adição de câmaras e apresenta informações do dispositivo. Contém ainda um painel de navegação lateral, no menu inicial, para listar câmaras e apresentar informações da conta de utilizador. Existe um menu para configuração dos perfis de Media que dá acesso a cada perfil. Para um perfil de Media é possível consultar e alterar as configurações de áudio, vídeo, metadados, PTZ e *analytics*. Há ainda a opção de visualizar uma *stream* de vídeo de um perfil de Media.

3. SERVIDOR REST ONVIF

O serviço existente [25], [26] estava alojado num servidor HTTP Apache versão 2.2.22 no sistema operativo XUbuntu. Pretende-se atualizar para a versão mais recente do Apache. A atualização permite dar seguimento às adições do Apache e utilizar novas configurações recomendadas. Na nova instalação manteve-se o sistema operativo e instalou-se a versão 2.4.7 do Apache. Recorreu-se à documentação do Apache [27] para atualizar as configurações e alterar as anteriores de modo a ter compatibilidade, melhorar a legibilidade e eliminar redundância. Este capítulo explica o processo de instalação do servidor Apache num novo local e as alterações e melhorias realizadas nas configurações já existentes.

Usou-se a extensão Postman [28] disponível para o *browser* Chrome para efetuar e testar os pedidos REST ao servidor. Esta extensão permite definir o tipo de pedido, cabeçalho e conteúdo de corpo da mensagem HTTP. Além disso, permite visualizar o conteúdo das respostas formatado em JSON.

O servidor usado está alojado na rede da Universidade do Minho, representado assim, um contexto real de utilização. O acesso remoto ao servidor foi feito através do Putty [29] que é uma ferramenta com a funcionalidade de terminal via SSH (Secure Shell), que é um protocolo para acesso remoto de forma segura e outros serviços de rede segura através de uma rede insegura [30]. Para transferir ficheiros via SSH recorreu-se ao WinSCP [31] que usa o SFTP (SSH File Transfer Protocol).

3.1 Estrutura de diretorias do Apache

De modo a entender o modo de funcionamento do Apache é necessário conhecer a sua estrutura, como está organizado o sistema de ficheiros, quais as funções dos diferentes locais e ficheiros de configurações.

O Apache tem uma diretoria base onde são feitas tipicamente as configurações do servidor. A diretoria de instalação é em "/etc/apache2/" e contém pares de subdiretorias com configurações disponíveis e as configurações ativas. Os pares de subdiretorias resumem-se essencialmente aos módulos, *sites* e configurações. A Figura 3.1 ilustra a estrutura de diretórios e os ficheiros da diretoria base do Apache.

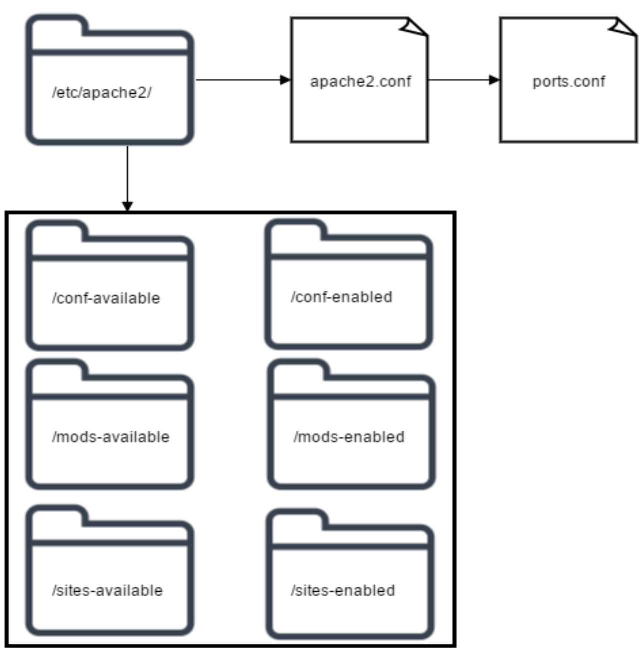


Figura 3.1 - Sistema de ficheiros do apache2

Os ficheiros da diretoria /etc/apache2/ são:

- **apache2.conf:** Este é o ficheiro de configuração principal do Apache. Ele agrupa várias partes incluindo todos os ficheiros restantes do Apache quando se inicia o serviço. Neste ficheiro são definidos parâmetros que influenciam o comportamento do servidor tais como diferentes parâmetros de *timeout,* nível de verbosidade dos registos, tipos de acessos entre outros.
- **ports.conf:** Contém as configurações das portas TCP. O Apache usa estas portas para esperar por mensagens HTTP. Este ficheiro permite habilitar e redirecionar as portas de modo a que todos os pedidos sejam encaminhados para os *sites* pretendidos.

As pastas da diretoria /etc/apache2/ são:

- **mods-available** e **mods-enabled:** contêm os módulos adicionais para serem executados no servidor Apache. Os módulos instalados encontram-se na pasta mods-available. Para um módulo ser usado pelo Apache tem de ser ativado, criando (com o comando a2enmod) um *symlink* na pasta mods-enable para os ficheiros de configuração [nome do módulo].conf e [nome do módulo].load.
- **conf-available** e **conf-enabled:** contêm configurações de módulos (e.g., módulos de segurança ou até mesmo um *web service*) disponíveis e ativas respetivamente. Para uma configuração/módulo ser usado pelo Apache tem de ser ativado, criando um *symlink* na pasta

conf-enable para os ficheiros de configuração [nome do módulo].conf e [nome do módulo].load.

- **sites-available** e **sites-enabled:** contêm os *sites* disponíveis e ativos respetivamente. Para criar um *symlink* na pasta sites-enabled é necessário ativar um *site* disponível em sites-available. Quando o apache é iniciado interpreta os ficheiros para quais os *symlinks* apontam, disponibilizando os *sites* ativos.

Quando é estabelecido um symlink para qualquer ficheiro de configuração, o ficheiro é ativo quando o apache é reiniciado.

3.2 Processo de configuração

Para executar o serviço num servidor apache é necessário efetuar um conjunto de passos. A aplicação destes é complementada com testes de funcionamento em cada passo devido a diferentes condições de instalação de um servidor. Devido a este facto é necessário ter conhecimentos de comandos e ferramentas de *debug* do apache que ajudam a concretizar os passos do processo de configuração corretamente. O apache tem os ficheiros “error.log” e “access.log”. para consultar erros e acessos respetivamente. O processo de configuração apache é o seguinte:

- **instalação do apache:** neste passo é instalado a versão mais recente do apache, à data. Para verificar se o apache foi corretamente instalado, inicia-se o serviço do apache (*service apache2 start*), e consulta-se localmente a página *web* por defeito através de um navegador. Esta página revela se existem problemas e exibe informações de utilização. As causas de erro na instalação podem ser exibidas quando se instala o apache. Caso não sejam dadas informações é necessário conhecer a estrutura de ficheiros do apache e o modo de funcionamento para verificar se as diretorias e ficheiros estão corretamente instalados. A partir daqui é necessário garantir que o sitio *web* tem configurações para guardar erros e acessos.

  Para imprimir o conteúdo dos ficheiros “error.log” e “access.log” usa-se os comandos “cat /var/log/apache2/error.log” e “cat /var/log/apache2/access.log” respetivamente.

  Para verificar o estado do apache usa-se os comandos:

  - **“apache2ctl status”:** fornece informações básicas sobre o estado do apache. Algumas informações de destaque são o estado, tempo de execução, ultimo reinício do

apache. Permite visualizar quantos *workers* em estado *idle* e a tratar de pedidos. Também apresenta um texto codificado para saber o que as *idle workers* estão a fazer.

  - **"apache2ctl fullstatus":** fornece informação extra em relação ao estado do Apache. O *fullstatus* dá informações sobre os pedidos que chegaram ao servidor.

- **geração de um certificado SSL *self-signed* para acesso seguro:** gera-se o certificado "restonvif.crt" assinado pelo próprio servidor (comando "make-ssl-cert /usr/share/ssl-cert/ssleay.cnf /etc/ssl/private/restonvif.crt") para implementar comunicação segura. Para verificar o funcionamento do certificado ativa-se o protocolo SSL e especifica-se o caminho do mesmo no ficheiro de configurações do sitio *web*. Nesta situação é necessário garantir que o certificado tem permissões de leitura, caso contrário, é exibido um erro no ficheiro "error.log". Depois, consulta-se a página por defeito utilizando o protocolo HTTPS e verifica-se o funcionamento da mesma. Como é um certificado gerado pelo próprio servidor, normalmente, os navegadores de acesso às páginas *web* não conhecem o certificado se este não for instalado, o que, gera um aviso com a opção de proceder na navegação, confiando no certificado.
- **configuração do sítio *web*:** faz-se novos ficheiros de configurações do sítio *web* para o serviço. Este passo consiste em fazer um ficheiro na diretoria "/etc/apache2/sites-available" para configurar o modo de acesso ao serviço *web,* tais como, definir comunicação segura e configurações de módulos. A verificação de funcionamento do serviço dependente dos passos seguintes uma vez que são acrescentadas configurações.
- **instalação do módulo fastcgi:** é instalado o módulo FastCGI para executar o serviço que consiste no ficheiro binário "umoc_rest.fcgi". Em adição são atualizados os ficheiros de configurações do sito *web* e do modulo FastCGI. Para verificar se o binário é colocado em execução corretamente consulta-se os ficheiros de *logs.* Através dos comandos "cat /var/log/apache2/error.log" imprime-se os erros, verifica-se se o binário iniciou execução e se existem dependências. As dependências precisam de ser instaladas com o comando de instalação, e.g., "apt-get install libssl". Para além dos erros, pode-se verificar se estão a ser feitos acessos corretamente com o ficheiro "access.log ".
- **iniciar/reiniciar o apache:** após completar a configuração do serviço inicia-se o apache com o comando "service apache2 start" e efetua-se pedidos para testar o funcionamento. O processo de consulta de erros é semelhante aos passos de cima, através da consulta de

estado do apache, ficheiros de *logs* e do conteúdo das respostas dos pedidos efetuados ao servidor.

Os passos realizados no processo de configuração do serviço são descritos no

Anexo VIII – Processo instalação servidor web REST. É descrito o modo de proceder em cada passo, e.g. comandos utilizados, e é feita uma descrição e justificação das diretivas usadas.

3.3 Renovação da configuração do serviço

Os ficheiros de configurações "ssl-rest-onvif.pt.conf" e "fastcgi.conf" foram renovados. As alterações foram as seguintes:

- remoção da configuração das páginas da aplicação web, uma vez que, não se faz uso das mesmas.
- substituição do controlo de acessos (atualização do controlo de acesso para mod_authz_host e remoção do controlo de acesso por htaccess).

O controlo de acesso na versão 2.2 é baseado no *hostname* do cliente, endereço IP e as características dos pedidos. É feito através de diretivas como Order, Allow, Deny e Satisfy. Na versão 2.4 o controlo de acesso é feito utilizando o módulo mod_authz_host que permite outras verificações de autorização. Assim, alterou-se as expressões antigas do controlo de acesso pelo novo mecanismo de autenticação. Em alternativa a compatibilidade com as configurações antigas é fornecida através do módulo mod_access_compat. Removeu-se a configuração para controlo de acessos através do ficheiro "htaccess" porque não é recomendada devido a fatores de desempenho e segurança.

As tabelas Tabela 3.1 e Tabela 3.2 mostram exemplos que comparam as configurações antigas com as mais recentes [32]. As configurações permitem respetivamente negar ou permitir todos os pedidos pretendidos. No servidor usa-se a diretiva "Require all granted" para aceitar pedidos ao servidor.

Tabela 3.1 - Configurações para negar todos os pedidos

Versão 2.2	Versão 2.4
Order deny,allow Deny from all	Require all denied

Tabela 3.2 - Configurações para permitir todos os pedidos

Versão 2.2	Versão 2.4
Order allow,deny	Require all granted

```
Allow from all
```

- remoção da linha AddHandler do ficheiro de configuração do serviço. Esta é redundante pois está configurada no ficheiro "fastcgi.conf".
- adição de registos de erros e acessos para *debug* e manutenção. As configurações usadas são as seguintes:

```
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
```

Usou-se a diretiva "ErrorLog" para registar erros no ficheiro de "error.log" e a "CustomLog" para registar informações de acessos no ficheiro "access.log".

- Removeu-se a configuração da diretiva "ScriptAlias /onvif /servidor/umoc_rest.fcgi" do módulo FastCGI porque está a ser feita no ficheiro do serviço. Esta diretiva associa o caminho "/onvif" para o binário "umoc_rest.fcgi".

As configurações finais dos ficheiros "ssl-rest-onvif.pt.conf" e "fastcgi.conf" podem ser consultados no Anexo I – Configurações do Servidor Web REST.

3.4 Funcionalidades adicionadas

O servidor está em constante desenvolvimento e evolução de modo a aumentar a quantidade de funcionalidades suportadas. Fez-se adições/alterações no servidor de modo a implementar/melhorar funcionalidades essenciais à biblioteca Java REST ONVIF. Pretende-se assim ajudar no desenvolvimento do servidor, disponibilizando novas funcionalidades e fornecer novas ideias de implementação para melhoria. Com isto, é possível desenvolver aplicações mais completas através do uso da biblioteca.

As adições ao servidor são as seguintes:

- Funcionalidade getServices para obter os serviços suportados por um dispositivo.
- Funcionalidade getServiceCapacilities de cada um dos serviços de um NVT para obter as *capabilities* respetivas.

O getServices é essencial pois permite obter os endereços dos serviços suportados além do Device Managment. Desta forma pode-se utilizar as funcionalidades dos outros serviços.

O getServiceCapabilities de um serviço oferece um modo de obter as capacidades do mesmo, permitindo ao programador desabilitar funcionalidades que não existam num dispositivo e utilizar apenas as suportadas.

Foram também feitas alterações transversais, que afetam toda a API, nomeadamente:

- Fornecer elementos JSON booleanos, que estavam a ser tratados como strings ou inteiros. Em JSON existem tipos booleanos sendo incorreto usar tipos diferentes.
- Correção de valores usados na serialização e desserialização incoerentes com a documentação.
- Acrescentar as entidades "dvcmgmt_capabilities", "deviceio_capabilities", "media_capabilities", "imaging_capabilities", "ptz_capabilities" e "dvcmgmt_services".
- Correção de problemas ao codificar URIs para o caracter "+". Codificou-se o caracter para UTF-8 com o valor "%2B" de acordo com a norma.

A resolução de problemas relativas ao mau uso de tipos JSON e incoerências da serialização/desserialização com a documentação é importante para que as aplicações cliente consigam usar a API do servidor corretamente. As entidades são adicionadas a enumerados que se associam a pedidos. Os enumerados para além de facilitarem a leitura do código são entidades em tempo de execução, ou seja, a sua informação é mantida.

Fez-se uma análise à API do serviço para sugerir alterações que melhorem o desempenho do servidor e facilitar a leitura da documentação da API. As respostas do serviço disponibilizam os dados relativos ao pedido ONVIF feito ao dispositivo e também os links que podem ser acedidos a partir desses dados, seguindo a filosofia REST de navegação em páginas. Desta forma, é enviada informação adicional nas respostas, que poderia ser evitada através da separação entre dados ONVIF e links da API. Para implementar esta divisão pode-se manter o modo como são feitos os pedidos ONVIF, retirando das respostas a informação relativa à API, colocando esta, nas respostas a pedidos que utilizam o método "OPTIONS". Este aspeto é importante porque o servidor tem o objetivo de diminuir o volume de tráfego, sendo que em diversos casos, o tamanho de dados da documentação do serviço é maior do que o conteúdo relativo ao ONVIF.

Por exemplo, para o serviço Device Managment para o método GET à URI "onvif/devicemgmt,193.136.12.31%2Fonvif%2Fdevice_service/SystemDateAndTime" tem-se a resposta da Figura 3.2 com o conteúdo (2) e a documentação (1). Recorrendo à mesma URI e usando

os métodos GET e OPTIONS conseguia-se obter duas respostas distintas uma com o conteúdo (2) e outra com a documentação (1).


```
"resources_uri" : [
   {
      "method" : "GET",
      "rel" : "_self",
      "title" : "System Date And Time",
      "uri" : "/onvif/devicemgmt,193.136.12.31%2Fonvif%2Fdevice_service/SystemDateAndTime"
   },
   {
      "method" : "PUT",
      "params" : [
       ...
      ],
      "rel" : "edit",
      "title" : "editar",
      "uri" : "/onvif/devicemgmt,193.136.12.31%2Fonvif%2Fdevice_service/SystemDateAndTime"
   }
],
"status" : {
   "msg" : "Sucesso!",
   "value" : 1
},
"time" : {
   "Day" : 15,
   "DayLightSaving" : false,
   "Hour" : 5,
   "Minute" : 34,
   "Month" : 7,
   "Second" : 10,
   "TimeFormat" : "Local time",
   "TimeZone" : "EGST+0:00:00",
   "Type" : "Manual",
   "Year" : 2016
}
}
```


(1)

(2)

Figura 3.2 - Documentação e conteúdo do pedido getSystemDateAndTime

4. BIBLIOTECA JAVA REST ONVIF

Neste capítulo é abordado o desenho da biblioteca Java REST ONVIF onde é descrito os requisitos, desenho da solução e implementação. O objetivo principal é a construção de uma biblioteca que abstraia a comunicação com o servidor REST. A biblioteca é desenvolvida em Java para Android e tem o objetivo de abstrair a API do serviço REST ONVIF permitindo às aplicações Android utilizar a API para comunicar com dispositivos NVT através do servidor *web* REST.

No capítulo 4.1 são especificados os requisitos da biblioteca Java REST ONVIF. O desenho da biblioteca é descrito no capítulo 4.2, o qual, contem analises e decisões de desenho. A descrição da implementação da biblioteca é realizada no capítulo 4.2.6 onde se pode consultar diagramas UML de vários componentes da biblioteca.

4.1 Requisitos

Neste capítulo é delineado os requisitos da biblioteca descrevendo e estabelecendo regras de desenvolvimento da mesma. A biblioteca interage com o servidor, implementa as funções da API do serviço e é utilizada pelo programador.

Requisitos

- Estrutura alinhada com o ONVIF.
- Configuração de câmaras ONVIF.
- Suportar os serviços possíveis para NVT
- Suportar de forma adequada os serviços opcionais, que as câmaras podem ou não implementar.
- Suportar os pedidos possíveis do servidor REST ONVIF.
- Utilizar comunicação segura com o servidor via HTTPS.
- Ter operações assíncronas e síncronas dos pedidos efetuados ao servidor REST ONVIF.
- Suportar os dados ONVIF em objetos Java.
- Fornecer informações de estado das operações, resultados e possíveis erros.
- Suporte para comunicação via serviço REST e direta via JNI
- Documentação Javadoc.

4.2 Desenho da API

A biblioteca é desenhada de modo a suportar a utilização de outras formas de comunicação alternativas ao serviço REST. Isto possibilita a integração de outros modos de comunicação. Pretende-se assim que a estrutura da biblioteca não sofra alterações com a adição de mais modos de comunicação.

A biblioteca tem 3 componentes principais, classes de dados, classes de serviço e classe OnvifCamera. Os 3 grupos de componentes estão ilustrados na Figura 4.1 num diagrama com classes. O diagrama não representa as relações entre elas nem o nível hierárquico pretendendo distinguir os tipos de classes existentes.

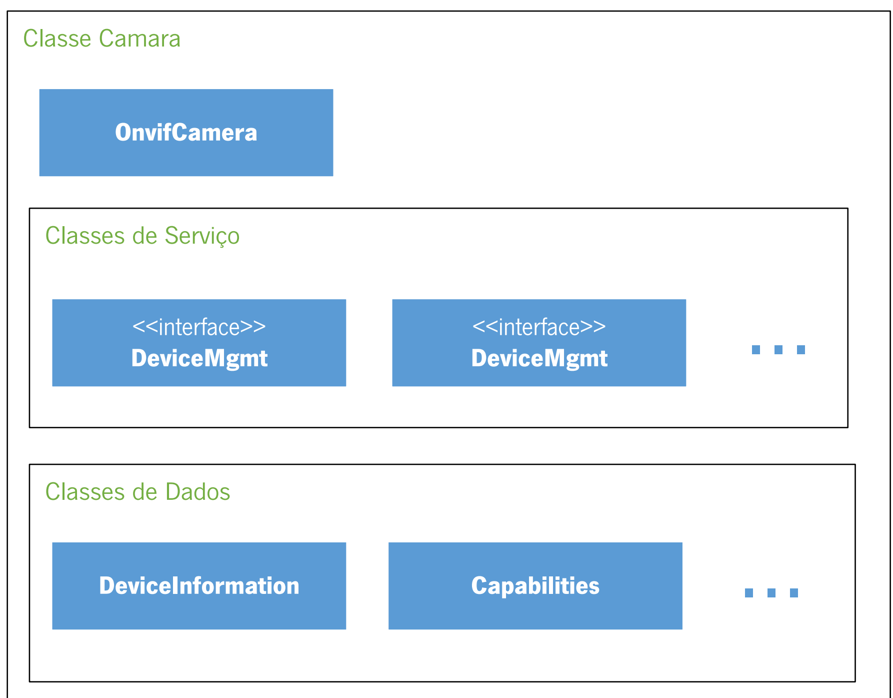


Figura 4.1 - Componentes principais da biblioteca Java REST ONVIF

As classes de dados são utilizadas pelos métodos das classes de serviço que lhes correspondem e servem para guardar dados relativos aos pedidos.

As classes de serviço ficam encarregues por realizar as operações de cada serviço para cada pedido ONVIF. Cada serviço implementa métodos síncronos e assíncronos. Os métodos utilizam a arquitetura de comunicação para realizar as operações ONVIF para posterior serialização/desserialização para/de JSON. As classes de dados podem guardar dados que se quer serializar/desserializar. Assim as classes de serviço podem criar/receber objetos das classes de dados para receber/enviar nas/nos respostas/pedidos.

A classe OnvifCamera disponibiliza as classes de serviços e gere os serviços implementados.

A implementação de métodos correspondentes às funções da biblioteca UMOC [3-4] é um requisito. Disponibiliza-se versões síncronas e assíncronas dos métodos de modo a proporcionar ao programador duas formas distintas de invocação de pedidos. Cabe ao programador decidir qual o tipo de método usar, sendo os métodos síncronos utilizados em operações sequenciais ou dependentes do resultado do pedido e os métodos assíncronos são utilizados para operações que não necessitem de resultados imediatos ou para tarefas em concorrência. Todos os métodos têm estas duas vertentes porque depende do contexto em que se utilizam e do que se pretende fazer com os pedidos.

4.2.1 Classe OnvifCamera

A classe OnvifCamera gere dispositivos, obtém, instancia e disponibiliza os serviços suportados. Esta classe utiliza as classes de serviço para definir os serviços de um dispositivo, e consequentemente as classes de dados para lidar com as informações acerca da câmara. A classe OnvifCamera é a classe de mais alto nível da API uma vez que abstrai os serviços implementados por um dispositivo. É responsável por guardar dados de acesso utilizador, *password* e endereço IP da câmara e:

- Tem como variáveis de instância objetos de serviço de um NVT;
- Cada variável serviço é do tipo “public final” para não poder ser alterada;
- As variáveis de serviços não suportados são nulas, sendo que, fornece um conjunto de métodos no formato supports<Service> para verificar quais os serviços suportados. Estes métodos são uma alternativa à verificação de nulidade dos objetos de serviço.
- Pertence ao *package* “org.uminho.onvif” que inclui os *packages* dos vários serviços, por exemplo, “org.uminho.onvif.media”.
- Tem dois construtores um com um parâmetro para o endereço do servidor REST e outro sem este parâmetro. Se se usar o endereço REST usa-se o modo de comunicação REST, caso contrário, usa-se o modo JNI. Depois o serviço pode ser instanciado com recurso a dois construtores respetivos às classes de serviço para REST e JNI. O desenho do diagrama de classes com a relação entre as classes do serviço PTZ e classe OnvifCamera é exibido na Figura 4.2. Assim, a classe OnvifCamera disponibiliza uma variável para o objeto da interface PTZ independente do modo de comunicação.

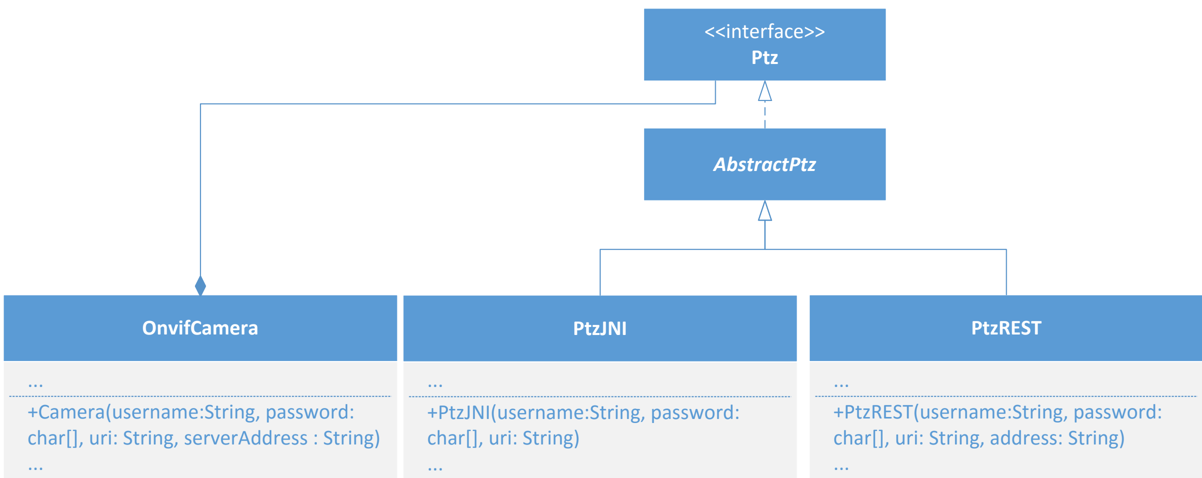


Figura 4.2 - Diagrama de classes para dois modos de comunicação

4.2.2 Classes de serviço

Devido à compatibilidade com diferentes modos de comunicação são definidas interfaces de serviço por cada serviço/*package*, que definem os métodos que são implementados por classes que implementam cada um dos modos. Assim, estas interfaces abstraem o modo de comunicação e têm:

- Nomes dos serviços (e.g., DeviceMgmt) definidas nos respetivos *packages* de serviço;
- Existem 2 métodos *overloaded* por cada uma das operações ONVIF, um síncrono e um assíncrono;

As classes de serviço implementam as interfaces, havendo duas delas por cada serviço/interface:

- Uma abstrata com o nome do serviço precedido do prefixo "Abstract" (e.g., AbstractDeviceMgmt), e contém a implementação dos métodos assíncronos. Esta classe não depende do modo de comunicação uma vez que apenas implementa a interface assíncrona para a invocação dos métodos síncronos;
- Outra concreta com o nome do serviço precedido do prefixo "REST", que deriva da primeira e implementa os métodos síncronos. Esta classe depende do modo de comunicação, pois os seus métodos são os que fazem os pedidos ONVIF.

Assim ficam definidas 3 classes de serviço (Interface, classe abstrata e classe correspondente ao modo de comunicação) numa hierarquia de implementação e herança com os seguintes requisitos:

- Ser final, para não poder ser derivada;
- Contém variáveis publicas só de leitura para guardar as *capabilities* respetivas;
- O construtor tem acesso *public*, para que a classe OnvifCamera possa instanciar os serviços. Consequentemente o programador pode ignorar a classe OnvifCamera e utilizar a biblioteca ao nível da camada dos serviços. Esta possibilidade evita a instanciação de serviços que o programador não quer utilizar;
- O construtor tem como parâmetros o endereço do serviço respetivo, um *username* e uma *password*, não há um construtor sem credenciais uma vez que quase todos os serviços têm apenas uma operação não autenticada, o GetServiceCapabilities, e esta não integra a interface;
- Os métodos verificam se são suportados pela câmara atual, e caso não sejam lançam uma exceção UnsupportedOperationException.

O diagrama de classes para as classes de serviço é exibido na Figura 4.3.

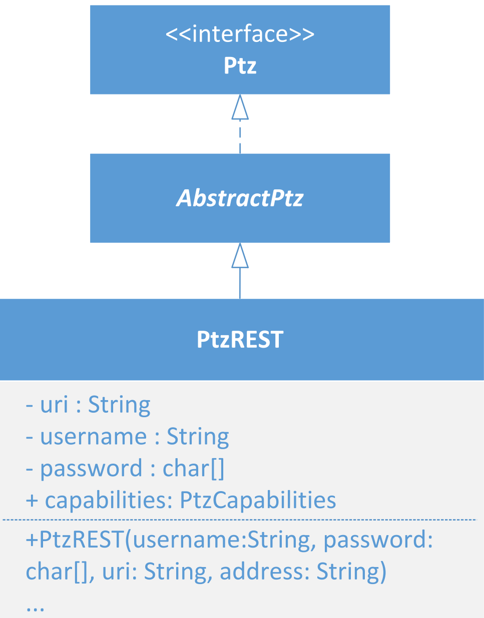


Figura 4.3 – Diagramas de classes de serviço

Os métodos assíncronos utilizam os métodos síncronos para efetuar os pedidos tendo a particularidade de executarem operações em paralelo. Decidiu-se utilizar a AsyncTask de Java para Android para fazer os métodos assíncronos porque permite executar operações de rede em *background* com recurso a *handlers* já definidos. Os resultados da AsyncTask podem ser publicados diretamente na UI Thread [33]. Além disso a AsyncTask fornece um meio de executar o verdadeiro *multithreading* [34] através de um parâmetro do tipo Executor. Este modo de *multithreading* permite definir se as invocações simultâneas de pedidos são sequenciais. Uma restrição da AsyncTask é que só

pode ser lançada a partir da UI Thread, no entanto, a versão síncrona dá toda a flexibilidade ao programador, pois pode ser executada a partir de qualquer outra *thread*. O Android disponibiliza métodos que permitem as *threads* publicarem resultados na UI Thread para os casos em que não se usa a AsyncTask [35].

Métodos síncronos

Os métodos síncronos realizam tarefas de rede diretamente para obrigar as operações a esperarem pelo valor de retorno. Por isso, estes métodos não necessitam de usar *threads* ou AsyncTasks para realização das tarefas de rede. Estes métodos são úteis em situações que é necessário obter determinadas informações antes de proceder para outras tarefas. No entanto, A UI Thread do Android não permite a execução de tarefas de rede diretas. Para isso é necessário utilizar uma *thread* para invocar e esperar pela realização das tarefas do método, ou recorrer a um AsyncTask. Estes fornecem uma interface transparente (as restrições são todas definidas na interface), que permite toda a flexibilidade ao programador em termos de invocação *multithreading*. Os métodos que obtêm dados só precisam do conteúdo JSON sendo que utilizam o processo de desserialização e os métodos que enviam informação necessitam de objetos das classes de dados para o processo de serialização. Como estes métodos utilizam operações de rede têm que ser invocados na UI Thread com recurso a uma AsyncTask ou a uma nova *thread*.

Os métodos síncronos têm as seguintes características:

- Os do tipo "get" (correspondentes aos verbos GET e DELETE), retornam uma instância da classe de dados respetiva, com os dados obtidos. Os do tipo "set" (POST, PUT) são geralmente void, pois há casos raros que têm retorno, como por exemplo, é o caso do *reboot* que devolve o tempo até ao reinício;
- Lançam uma exceção OnvifException quando recebem um erro reportado pelo serviço. Esta exceção é específica da biblioteca (definida no *package* "org.uminho.onvif") e deriva da classe Exception, ou seja, uma exceção *checked* [36], porque todos esses erros têm que ser tratados pela aplicação. Esta exceção tem a mensagem de erro recebida no JSON da API REST.

Assim, os métodos síncronos declaram-se com os formatos "DateType syncGetMethodName() throws OnvifException;" e "ReturnType syncSetMethodName(DateType dateType, ...) throws OnvifException;". O DateType representa uma classe de dados.

Métodos assíncronos

Os métodos assíncronos utilizam uma AsynTask [33] e portanto, têm que ser invocados na UI Thread, mas fornecem uma interface de mais alto nível (*event-driven*, e sem exceções). Têm as seguintes características:

- Têm o retorno do tipo void;
- Têm um parâmetro adicional que é um objeto *listener* para tratar o resultado (i.e., receber os dados obtidos, lidar com os erros ocorridos e atualizar o progresso da operação).
- Usam uma AsyncTask para invocar os métodos síncronos.

Assim, os métodos assíncronos declaram-se com o formato "void asyncGetMethodName(...)" e "void asyncSetMethodName(DateType dateType...)". O DateType é um tipo de uma classe de dados.

A principal vantagem dos métodos assíncronos é que permitem invocações diretas na UI Thread. Além disso têm o *multithreading* simplificado e funcional com *handlers* de erro, progresso e resultado que são úteis ao programador. As tarefas assíncronas não obrigam a esperar pelo resultado o que pode ser útil para situações em que se quer atualizar valores da UI sem interromper o trabalho do utilizador. Um exemplo é a atualização das informações da data e hora enquanto o utilizador consulta outros dados.

Listeners

Os métodos do tipo "set" apenas necessitam dos eventos onError() e OnProgress(), por isso, define-se uma interface única com esses mesmos métodos, denominada OnvifListenerNoResult. Cabe ao utilizador gerir a forma de distinguir os resultados de métodos invocados em sequência, seja utilizando diferentes objetos *listener* ou mantendo estado num objeto partilhado pelas várias invocações.

Para os métodos do tipo "get" a questão é mais complexa. Na possibilidade extrema de se definir uma interface única, comum a todas os métodos, o método onResult() teria que ter um parâmetro do tipo Object para ser compatível com qualquer tipo de dados obtidos. Esta abordagem simplifica a biblioteca, mas requer que a aplicação faça o *cast* do argumento numa sequência *if-else-if* com condição *instanceof*, o que tem problemas de desempenho, manutenção do código e não é Orientado-a-Objetos (OO). Outro problema é a falta de clareza da interface pois é preferível ter o tipo de retorno explícito ao invés de Object que não é informativo.

Para evitar os problemas acima pode ter-se uma interface por cada método “get” de cada serviço. Esta abordagem está no extremo oposto da anterior, gerando uma geração de interfaces que complicaria a API da biblioteca (em vez de uma série de if-elses na implementação).

A solução encontrada consiste em definir uma interface genérica OnvifListener<Result>, derivada de OnvifListenerNoResult, que define o método onResult e herda os métodos onError e onProgress. Esta é uma opção que evita criar inúmeras interfaces, evita ter como tipo de retorno Object e tem o tipo Result que pode ter o valor de retorno pretendido para cada método “get”. Desta forma tem-se uma abordagem que combina OO com programação genérica, que resolve todos os problemas de desempenho, manutenção, flexibilidade e facilidade de utilização.

Esta é a solução escolhida e cada interface tem as seguintes características:

- Declaração do método do género “Result onResult()”;
- Argumento do método onResult() igual ao tipo de dados obtido;

Supondo que se pretende realizar um método com o tipo de retorno “MediaProfile” o *listener* é declarado da seguinte forma: “OnvifListener<MediaProfile> listener”. Assim o método do *listener* é declarado da forma “onResult(MediaProfile mediaProfile)”. O programador tem um *listener* com os *handlers* disponíveis para reescrita e com o tipo de retorno especificado nos métodos assíncronos que o utilizam. Pode-se reescrever o método onResult(MediaProfile mediaProfile) para trabalhar com o objeto de dados mediaProfile recebido pelo *listener.*

Os *listeners* precisam de receber os eventos onError, onResult e onProgress. O *listeners* OnvifResultListener, OnvifErrorListener e OnvifProgressListener têm os respetivos métodos:

- public void onResult(<TipodosDados> d), para receber os dados no caso de métodos do tipo “get”. Caso o método assíncrono não tenha tipo de retorno, usa-se um *listener* sem este método;
- public void onError(String description), invocado em caso de erro com a respetiva descrição; este método é imprescindível uma vez que a AsyncTask tem que apanhar todas as exceções;
- public void onProgress(int level) para informar sobre o avanço da operação de uma forma simplificada em percentagem.

Estes métodos são definidos em interfaces para poderem ser implementadas por qualquer classe da aplicação, oferecendo assim toda a flexibilidade.

4.2.3 Classes de dados

As classes de dados guardam informações de configuração das câmaras. Estas são simétricas ao tipo de conteúdo estruturado num objeto JSON dos pedidos e respostas do serviço *web* REST. Os objetos destas classes são utilizados nos métodos e *listeners* das classes de serviço. Existe uma classe para cada estrutura de dados ONVIF com os seguintes requisitos:

- Pertence ao *package* do respetivo serviço; as que são partilhadas por serviços são definidas num só: aquele que de acordo com a classificação NVT seja obrigatório ou mais importante;
- Ser final, para não poder ser derivada
- Ter variáveis que são uma representação Java dos dados ONVIF da API do serviço.
- As classes que correspondem a dados apenas obtidos das câmaras (ou seja, nunca são utilizados como parâmetros de entrada) têm as seguintes características:
    a. Os construtores das classes que correspondem a dados obtidos diretamente das câmaras (ou seja, nunca são utilizados como parâmetros de entrada) têm acesso do nível *package*; assim só podem ser criados objetos dessas classes pelas classes do *package* do próprio serviço, concretamente os métodos da classe serviço que são Factory Methods [37] desses objetos;
    b. as variáveis são *final*, para que não sejam alteradas (i.e., sejam *read-only*).

A biblioteca UMOC existente tem estruturas de dados e funcionalidades que a biblioteca em Java deve implementar. Na análise da biblioteca obtiveram-se regras para a conversão dos tipos de variáveis entre da biblioteca UMOC e a Java. Os dados são geridos através de estruturas de C, pelo que, se concebeu classes de dados através de classes Java. As regras adotadas para o desenvolvimento das classes em relação às estruturas são:

- Os elementos opcionais são do tipo apontador, pois podem ou não ir/vir no/na pedido/resposta JSON. Opta-se por usar objetos invés de tipos primitivos de Java;
- São usados nomes completos nas classes (por exemplo MetaDataConfiguration) e nomes abreviados ou idênticos nas variáveis membro (e.g., mdConf ou metaDataConfiguration).
- O tipo char* passou a String; com a exceção das variáveis para *passwords* que devem ser char[] [38] devido a fatores de segurança;
- Os pares "int sizeX" e "struct tipoX *X" são convertidos em ArrayList<tipoX>. O motivo é que a classe MediaProfile é utilizada também no pedido setProfile, pelo que se quisermos adicionar

um filtro ao metadataconf não vamos copiar um *array* inteiro e acrescentar um novo elemento, é preferível utilizar um ArrayList que cresce dinamicamente;

- Os pares "int size" e "char** any", são para traduzir para um *array* de Strings Java, pois os *arrays* têm a propriedade *length* (e.g., "um Array.length") que permite saber o seu tamanho;
- Removeu-se os prefixos "schema__" de alguns tipos de estruturas (e.g., schema__ItemList deve ser ItemList, e schema__Config deve ser Config).

4.2.4 Serialização e desserialização JSON

As classes de dados são instanciadas com informações das respostas ou enviadas nos pedidos. Os pedidos/respostas usam o formato de dados estruturados JSON na troca de mensagens. O processo de serialização faz a conversão das classes de dados para JSON e a desserialização converte o conteúdo JSON para as classes de dados. O processo de serialização e desserialização é feito ao nível das classes de serviços nomeadamente nos métodos síncronos.

4.2.5 Desempenho

Java é uma linguagem de alto nível orientada a objetos o que requer a criação de objetos em programas convencionais. A gestão de memória não é tao controlada pelo programador como em linguagens de baixo nível como C, o que pode representar redução de desempenho em algumas situações. Existem inúmeras maneiras de se atingir um resultado computacionalmente, mas nem todas são as melhores a nível de desempenho. Assim, é necessário aplicar boas práticas de programação que serão descritas nesta subsecção.

Foram definidas algumas regras de programação para conceber uma biblioteca otimizada. As regras criadas são baseadas em duas regras simples:

"Don´t do work that you don´t need to do.

Don´t allocate memory if you can avoid it." [39]

O que estas regras transmitem é que não se deve utilizar operações desnecessárias e alocar memória que pode ser evitada.

As regras adotadas e razões são as seguintes:

- Evitar a criação de objetos desnecessários porque esta requer alocação de memória significativa. Deste modo, evita-se alocar memória desnecessária [39], [40], [41].

- Usar métodos static ou métodos da classe em vez de métodos do objeto quando não é necessário o acesso direto aos campos. As invocações são mais rápidas e é uma boa prática uma vez que chamar o método não altera o estado do objeto [39].
- Usa-se "static final" na declaração de qualquer constante que utilize tipos primitivos do Java. O acesso às constantes é feito diretamente e a instrução de procura é mais barata a nível de desempenho.
- Evitar a utilização de *getters* e *setters*. Em Android a chamada de métodos virtuais é relativamente cara. Pode-se ter *getters* e setters numa interface pública como uma boa prática de programação orientada a objetos, mas dentro de uma classe deve-se fazer o acesso direto aos campos. Aceder a um campo diretamente é 7 vezes mais rápido que invocar um *getter* [39].
- Quando possível, escolher o ciclo *for-each* em detrimento de outras sintaxes. Este ciclo evita o típico cálculo do tamanho do *array* quando necessário. Também é mais eficaz em dispositivos sem JIT (*just-in-time*), ou seja, dispositivos que em tempo de execução compilam um programa.

Estas regras aplicam-se na implementação, mas também influenciaram o desenho anteriormente apresentado.

4.2.6 Análise de exceções

As exceções que podem ocorrer advêm do NVT, servidor ou da própria biblioteca. Desta forma, fez-se uma análise dos tipos de erros que podem ocorrer e como os tratar na biblioteca.

As respostas podem ter erros que ocorrem no servidor ou no NVT, pelo que, é filtrada a descrição do erro contido na resposta do servidor e posteriormente convertido numa OnvifException do tipo *checked*. No caso de exceções gerados pelo código da biblioteca, é necessário analisar se ocorrem devido a fatores externos e se faz sentido convertê-los numa OnvifException. Se alguma exceção ocorrer devido a erros de implementação, naturalmente, não devem ser convertidos em exceções *checked* porque não são erros recuperáveis.

A ProtocolException [42] é um exceção *checked* que ocorre se o método não é suportado pela aplicação HTTP, pelo que, é gerada por erros de implementação ou incumprimentos de requisitos. Pode ser gerada por um erro TCP inerente ao protocolo sendo que erros de rede estão fora do controlo da biblioteca.

A IOException [43] é uma exceção *checked* que sinaliza um erro relacionado com IO nomeadamente que a operação falhou ou foi interrompida, como por exemplo quando falha a ligação de rede. Esta é uma situação com a qual a aplicação poderá lidar.

A JSONException [44] é uma exceção *checked* que ocorre pelas seguintes razões: tentar analisar ou construir documentos malformados, usar "null" como nome, usar tipos numéricos indisponíveis em JSON, tais como, não ser um número ou ser uma infinidade e pesquisa com índice fora do alcance ou de nome inexistente. Se esta exceção ocorre na serialização, então é apenas por erro de programação na aplicação cliente; se acontecer na deserialização, pode ser também por erro do servidor. A aplicação nada pode fazer para recuperar deste tipo de erros.

Alguns erros são controlados pela biblioteca pelo que é garantido que não ocorrem, e.g., antes de obter um objeto num JSONObject através de uma *tag* deve-se verificar se o JSONObject contem a *tag*. A OnvifExeception também gera erros devido à lógica da API do serviço *web*, uma vez que, as respostas têm um campo com o tipo de sucesso e com a descrição do erro caso exista. Cria-se uma OnvifException se determinadas operações não forem suportadas pela câmara. Alguns dos fatores analisados nas Exceptions podem ser prevenidos, como por exemplo, verificar a conetividade antes realizar uma operação de rede.

4.3 Implementação

Esta secção explica a implementação da biblioteca assim como justifica as opções tomadas. Os diagramas de classes e sequência respeitam as definições do UML 2, a versão atual à data. Alguns diagramas são apenas representativos não seguindo necessariamente um modelo universal de representação. Ao longo da implementação da biblioteca foram feitas melhorias ao desenho. É apresentada a versão final da implementação através da descrição dos seus componentes.

No desenvolvimento da Biblioteca Java REST ONVIF utilizou-se o Android Studio [45] que é um IDE com diversas funcionalidades para desenvolvedores Android. Este tem suporte e está em constante desenvolvimento pela Google com diversas funcionalidades para o programador O Android Studio ajuda no desenvolvimento de Java para Android e em testes de aplicações em emuladores ou dispositivos reais. Para além do emulador predefinido do Android Studio, usou-se, o Genymotion [46] e o Xamarin [47] para Android que são emuladores mais rápidos e com menor consumo de memória. O dispositivo utilizado para testes reais é um Nexus 5 da primeira geração.

As diversas componentes da arquitetura estiveram em constante análise, por isso, ao longo da implementação existiu várias alterações que permitiram chegar à solução final. Para usar formatos vetoriais na edição de diagramas usou-se a ferramenta Microsoft Visio para criar, desenvolver e atualizar os mesmos. Assim, os diagramas ficam num formato conhecido e associados a uma ferramenta própria, e com compatibilidade para o desenho de diagramas UML.

A Figura 4.4 é o diagrama de *packages* da solução final da biblioteca com todos os *packages* e a Figura 4.5 é o diagrama de classes do *package* base com as classes auxiliares para a biblioteca. A biblioteca está no package "org.uminho.onvif" que contém algumas das classes mais importantes e os *packages* dos serviços. As classes de dados e classes de cada serviço estão localizadas nos *packages* respetivos aos seus serviços. As classes de ajuda e comuns aos serviços estão no *package* base assim como a classe OnvifCamera.

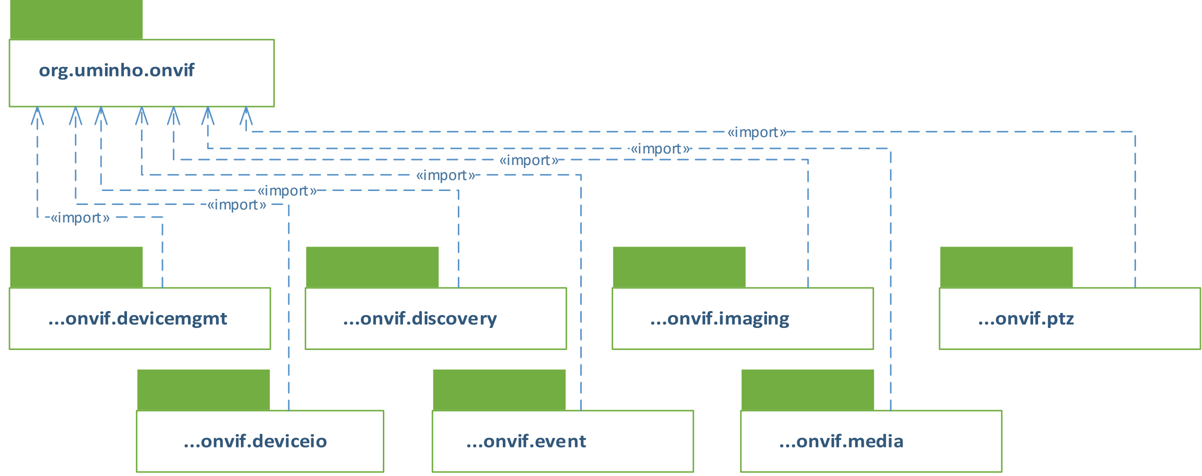


Figura 4.4 - Diagrama de *packages* da Biblioteca REST ONVIF

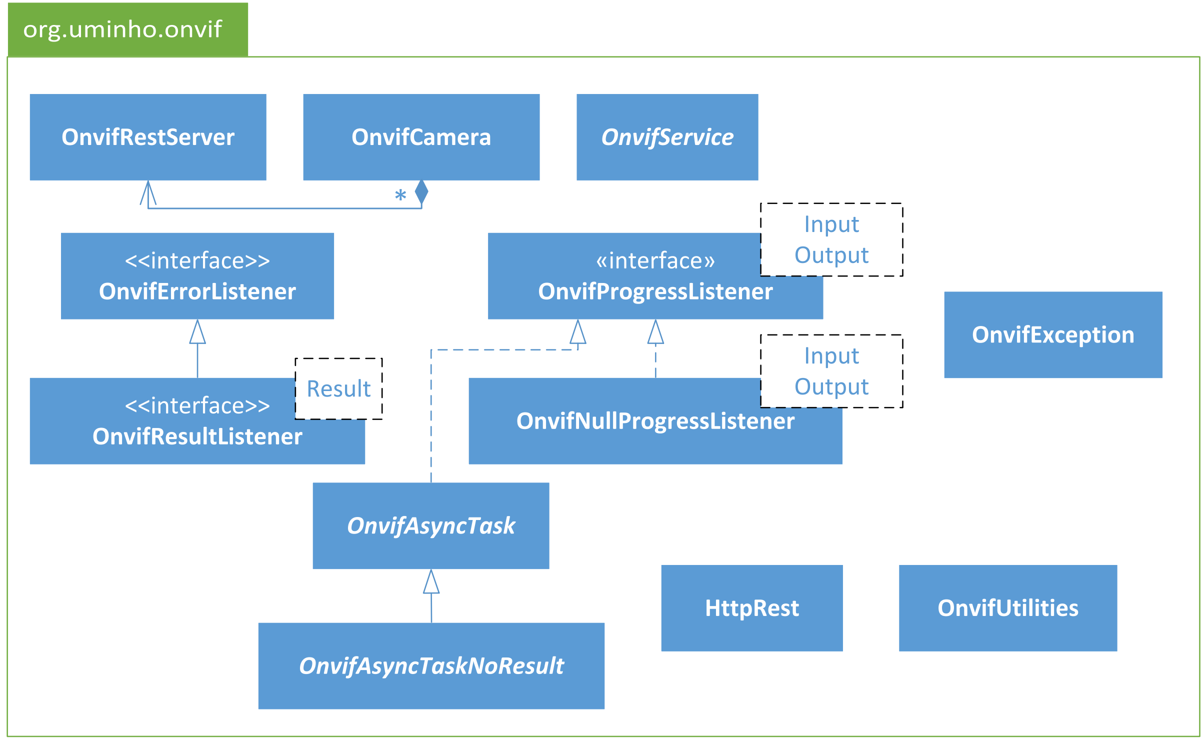


Figura 4.5 - Diagrama de classes do *package* org.uminho.onvif.

No *package* de base encontram-se no mesmo pacote, os *handlers* do resultado, progresso e erro a serem utilizados pelos programadores. As restantes classes só são usadas internamente pela biblioteca e não necessitam de ser conhecidas pelo programador.

A principal classe do *package* base é a classe OnvifCamera que obtém os serviços implementados pelo dispositivo e instancia os respetivos objetos de serviço. Para instanciar os objetos serviço é usado o método síncrono getServices.

4.3.1 Classes de utilidades

A classe de ajuda é para uso interno na API e para fornecer ao utilizador métodos úteis na utilização da biblioteca. O uso desta classe, por parte do programador, é opcional sendo que a classe facilita no desenvolvimento de aplicações. Todos os métodos que só devem ser usados pela própria biblioteca não estão visíveis ao programador.

A classe de ajuda, “OnvifUtilities”, tem métodos que são usados pelas classes de serviço e métodos caso o programador necessite de usar certificados assinados pelo próprio servidor. Os métodos são do tipo *static* para serem acedidos diretamente através da classe. O diagrama da classe OnvifUtilities com os métodos disponibilizados é exibido na Figura 4.6.

OnvifUtilities
+throwsOnvifException(msg: String) : void +throwsOnvifException(json: JSONObject): void +hasConnection(context : Context): boolean +getNanoTime(): String +trustAllHost() +trustHost(hostname : String, rRawCrt : Integer)

Figura 4.6 - Diagrama da classe OnvifUtilities

Os principais métodos da classe OnvifUtilities são:

- throwsOnvifException(String msg), para lançar um OnvifException com determinada mensagem. É usada na biblioteca quando se conhece diretamente a mensagem de erro que se quer usar na *exception*.
- throwsOnvifException(JsonObject json), recebe o objeto JSON de uma resposta do servidor. Caso o conteúdo da resposta indique erro no processamento do pedido é lançado uma OnvifException. É usada essencialmente nos métodos síncronos quando se recebe uma resposta e invoca o método "throwsOnvifException(json)" quando se obtém o objeto json.
- trustAllHosts(), para confiar em todos os *hosts* a que se conecta. Este método é pouco seguro pois não há restrições nas ligações que se podem estabelecer. No entanto não é preciso ter o certificado para utilizar este método. Este método usa-se na aplicação Android, não sendo invocado na biblioteca.
- trustHost(String hostname, Integer rRawCrt), para confiar num determinado *host* de modo a estabelecer uma ligação segura. Este método requer a importação do certificado.

4.3.2 Classe OnvifHttpREST

A biblioteca necessita de um modo de efetuar pedidos ao servidor REST e receber a resposta em JSON através de uma camada de HTTP. A camada de comunicação necessita de respeitar o formato de pedidos definidos na API do serviço REST que inclui quatro tipos de verbos, URL, conteúdo do cabeçalho e de corpo. O servidor REST instalado utiliza comunicação cifrada através do protocolo HTTPS e utiliza um certificado assinado pelo próprio servidor, sendo que, é preciso suporte para comunicação HTTPS. Assim desenhou-se uma classe de comunicação que tem as seguintes características:

- Disponibiliza métodos que implementem todos os verbos HTTP suportados pelo serviço *web* REST. Os métodos auxiliares são privados;
- Suporta o envio/receção de dados em formato JSON no *body* HTTP.
- Os parâmetros das credenciais de acesso para autenticação são inseridos no cabeçalho do pedido e são fornecidos pelos métodos síncronos.
- Comunicação segura/cifrada por HTTPS.
- Tem visibilidade pública para ser acedida pelas classes de serviço;
- Não tem construtores e todos os métodos são estáticos, fornecendo um acesso direto aos métodos públicos sem a necessidade de criar objetos adicionais, o que, reduz o consumo de memória;
- Tem um enumerado interno HttpRequestType que define os 4 tipos de métodos necessários.

A camada de comunicação consiste na classe OnvifHttpRest localizada no *package* "org.uminho.onvif". A estrutura interna da classe de comunicação pode ser modificada futuramente sem que influencie as camadas superiores. Para implementar a comunicação utilizou-se a API do *package* "java.net" [48]. Esta API inclui o cliente HttpURLConnection, que tem suporte para HTTPS, *streaming, uploads*, *downloads, definições de timeout,* IPv6*.* Descartou-se a possibilidade de utilização do "Apache HTTP cliente" [49] porque foi descontinuado para a versão 6.0 do Android e pretende-se desenvolver uma aplicação para a versão mais recente de preferência com suporte para as tecnologias utilizadas.

O diagrama da classe OnvifHttpRest é exibido na Figura 4.7 permite observar os 4 métodos principais públicos para se usar na API. Os métodos disponibilizados que têm operações de CRUD têm os parâmetros "username" e "password" usados para autenticação, URL que especifica o caminho *web*, o "progressListener" usado na atualização do progresso e o "content" usado para envio de informação no corpo do pedido. Os métodos privados não estão especificados no diagrama de classes.

HttpRest
+get(url:String, username:String, password:String, progressListener:OnvifProgressListener): JSONObject +delete(url:String, username:String, in password:String, progressListener:OnvifProgressListener): JSONObject +post(content:JSONObject, url:String, username:String, password:String,progressListener:OnvifProgressListener): JSONObject +put(content:JSONObject, url:String,username:String, password:String, progressListener:OnvifProgressListener): JSONObject

Figura 4.7 - Classe HttpRest para comunicação HTTP

A classe HttpRequestType é um enumerado, interna à classe HttpRest, que enumera os 4 tipos de verbos HTTP. Os valores possíveis do enumerado são GET, POST, PUT e DELETE no formato de

uma *string*. Escolheu-se este formato uma vez que é necessário obter o valor do HttpRequestType em *string* para enviar no cabeçalho do método "setRequestMethod(...)" da classe HttpURLConnection. Assim, a OnvifHttpClient tem um enumerado para permitir que existam métodos privados que unificam mais funcionalidades e evitam repetição de código.

Existem alguns passos essenciais efetuados pelo OnvifHttpRest quando se invoca um dos métodos para fazer um pedido. Os passos estão representados no diagrama de sequência da Figura 4.8. O primeiro passo consiste na obtenção da conexão para dar início à comunicação. Com a conexão aberta é preparado o cabeçalho com o tipo de pedido a efetuar, dados do cabeçalho, tipo *encoding* de dados e se existe informação para enviar. Para os métodos "put(params)" e "post(params)" existe envio de informação sendo que estes métodos tem o passo adicional de enviar dados JSON no corpo do pedido. Neste passo recebe-se a resposta do servidor que é convertida para JSON.

Como o servidor utiliza comunicação segura para cifrar os dados utiliza-se a classe HttpsURLConnection invés da HttpURLConnection. Para certificados que não são assinados por uma entidade certificadora é necessário definir se são confiáveis de modo a ser permitido efetuar a ligação. A classe OnvifUtilities tem métodos para confiar em um/qualquer certificado.

Os métodos privados de comunicação unificam o código em métodos do tipo "get" e "set". Os métodos "get(params)" e "delete(params)" são considerados do tipo "get" porque não têm conteúdo de corpo enquanto os métodos "post(params)" e "put(params)" são considerados do tipo "set" por terem conteúdo de corpo. O diagrama de sequência da Figura 4.8 corresponde ao processo de operações de um pedido sem *content*. (ou seja, um pedido do tipo *get*). O pedido usa o verbo GET, sendo que, para o DELETE apenas muda o nome do método.

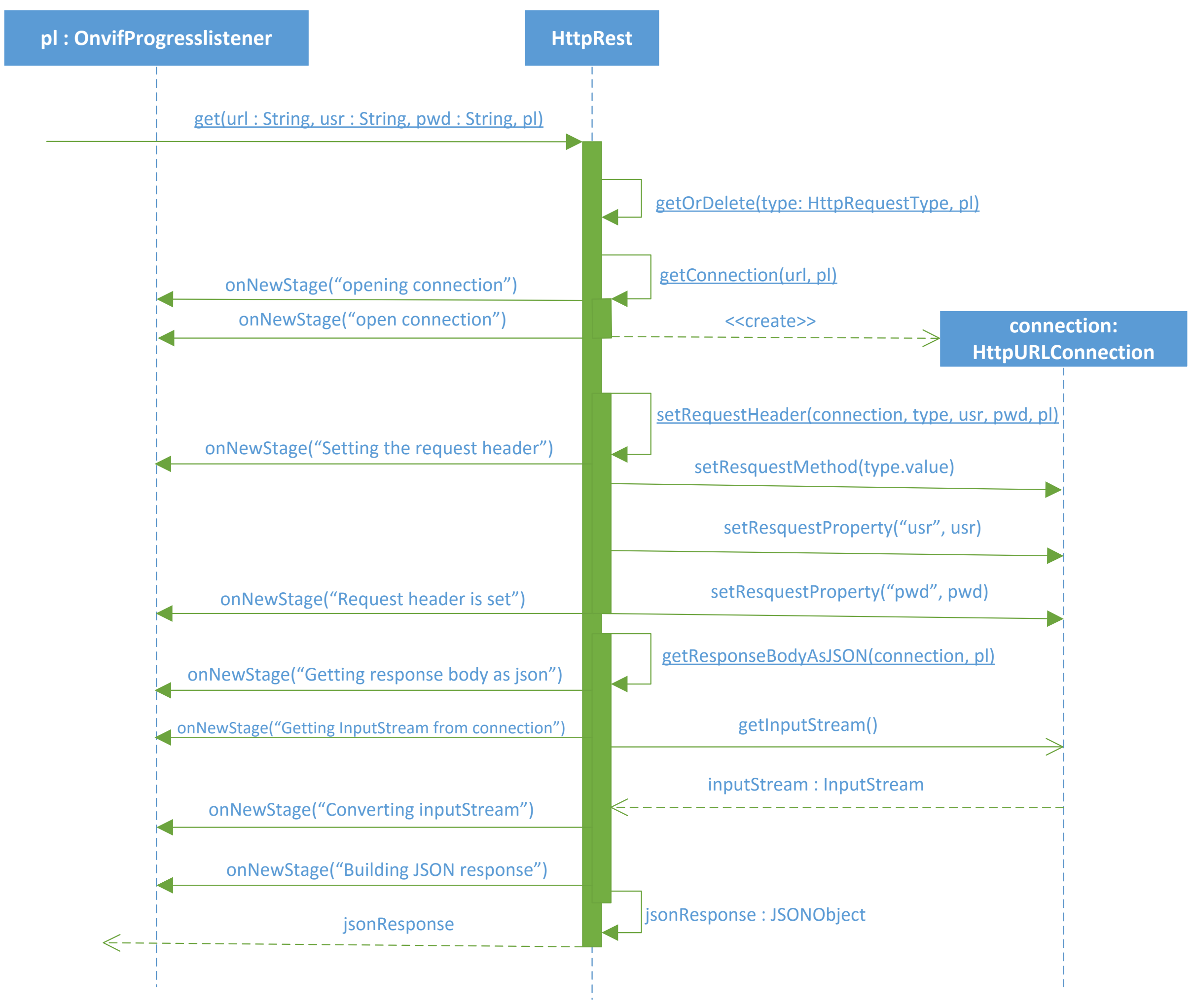


Figura 4.8 - Diagrama de sequência de um pedido *get*

O diagrama sequência da Figura 4.9 é para um pedido com *content* (ou seja, um pedido do tipo *set*). O pedido usa o verbo POST, sendo que, para o PUT apenas muda o nome do método.

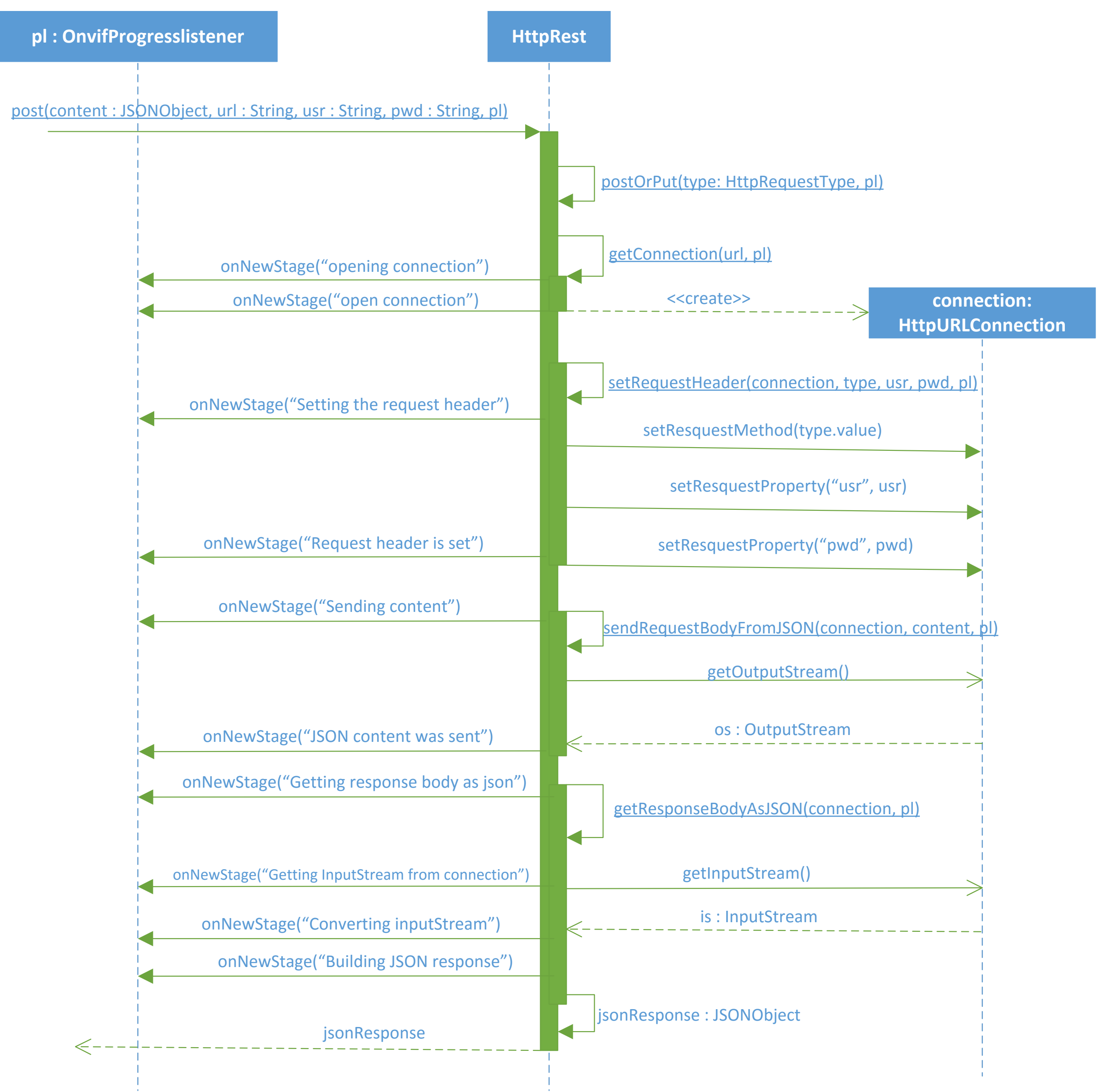


Figura 4.9 - Diagrama de sequência do pedido *post*

As operações de um pedido "get" e "set" são semelhantes. Inicialmente após a invocação de um método *get/post/put/delete* é necessário estabelecer a conexão através do URL. De seguida é necessário preencher o conteúdo de cabeçalho com o tipo de verbo HTTP e com as credenciais da câmara. Caso seja necessário conteúdo de corpo, para os métodos do tipo "set", é preenche-se o corpo do pedido através da conexão estabelecida. Por fim obtém-se o inputStream que é convertido em JSON para ser retornado pelo método inicial. O *listener* de progresso é usado neste processo de modo a fornecer informações de estado do pedido para as várias fases da operação. Para atualizar o progresso com uma determinada mensagem invoca-se o método "onNewStage". Os diferentes tipos de mensagens usados na classe de comunicação estão representados nos diagramas de sequência.

4.3.3 Serialização e desserialização JSON

A serialização de dados é usada nos métodos do tipo POST e PUT para enviar informação JSON no corpo do pedido HTTP. O conteúdo das respostas está no formato JSON, sendo que todos os métodos necessitam de desserializar dados. No processamento das mensagens optou-se por utilizar bibliotecas Java dedicadas à serialização e desserialização JSON. A escolha da biblioteca baseou-se na sua simplicidade e capacidade de lidar com tipos primitivos de Java, objetos e *arrays* JSON. A biblioteca escolhida usa o *package* de Java "org.json" e tem classes como JSONObject e JSONArray para criar objetos e *arrays* JSON. Estas classes disponibilizam métodos para serializar/desserializar tipos primitivos de Java, *arrays* e objetos, fornecendo assim, um meio para criar ou obter qualquer estrutura JSON.

Na serialização/desserialização aplicam-se as seguintes regras:

- Usar a classe JSONArray para guardar/criar *arrays* JSON.
- Usar a classe JSONObject para guardar/criar objetos JSON.
- Usar os *getters* e *setters* fornecidos pelas classes JSONObject e JSONArray para colocar/retirar valores que podem ser tipos primitivos de Java, *arrays* e objetos JSON.

Os métodos GET obtêm informações sobre a câmara e de estado da operação, enquanto os restantes métodos têm informação de estado da operação. As classes de serviço utilizam as classes de dados para estruturar informação, sendo estas, usadas para guardar informação desserializada e para receber objetos de dados que se pretende serializar. Assim é necessário converter objetos de dados para objetos JSON e vice-versa.

Na serialização dos dados JSON, a técnica a utilizar, passa por criar um JSONObject onde se irá construir o objeto JSON pretendido. Na desserialização recebe-se um objeto JSON (JSONObject) que deve ser traduzido num objeto de dados.

Os objetos JSON usam pares de valores no formato nome (único) e valor, em que os valores podem ser objetos, *arrays* e tipos primitivos. Um *array* pode ter valores como objetos, *arrays* e tipos primitivos. Estas são as funcionalidades necessárias para lidar com a serialização e desserialização nos métodos síncronos.

Na serialização o objeto JSON é construído a partir dos objetos de dados invocando os métodos JSONObject.put(...) em sequência por cada membro de uma classe de dados e o JSONArray.put(...) em ciclo por cada elemento de um *array* que se quer inserir. Os opcionais requerem uma condição para verificar se existem no objeto de dados antes de serem inseridos no objeto JSON.

Para enumerados é possível retirar diretamente o valor que se pretende através do acesso ao *value* definido no enumerado e inserir num objeto JSON. Por exemplo, para obter uma *string* do enumerado UserLevel usa-se a expressão UserLevel.value.

Na desserialização o processo é semelhante mas com recurso aos métodos get(…) do JSONObject e JSONArray. Para dados opcionais usa-se o método has(…) que verifica a existência do valor opcional no objeto JSON, não sendo necessário comparar com *null* (objetos de dados opcionais podem ser *null*). Os objetos de dados do tipo enumerado são instanciados com o seu método "valueOf(value)" em que *value* é o valor recebido em JSON. Por exemplo para instanciar um enumerado do tipo UserLevel usa-se a expressão "UserLevel.valueOf("admin")".

A Figura 4.10 representa a equivalência entre um objeto de dados e objeto JSON para uma coleção de scopes. É possível verificar no objeto JSON existe um *array* com o nome "scopes" com uma lista de objetos. Os objetos do *array* estão associados à classe ScopeInformation do objeto de dados e o *array* está associado ao ArrayList<ScopeInformation>. Os valores do objeto JSON "scope" e "type" correspondem ao campo *scope* e *type* da classe ScopeInformation.

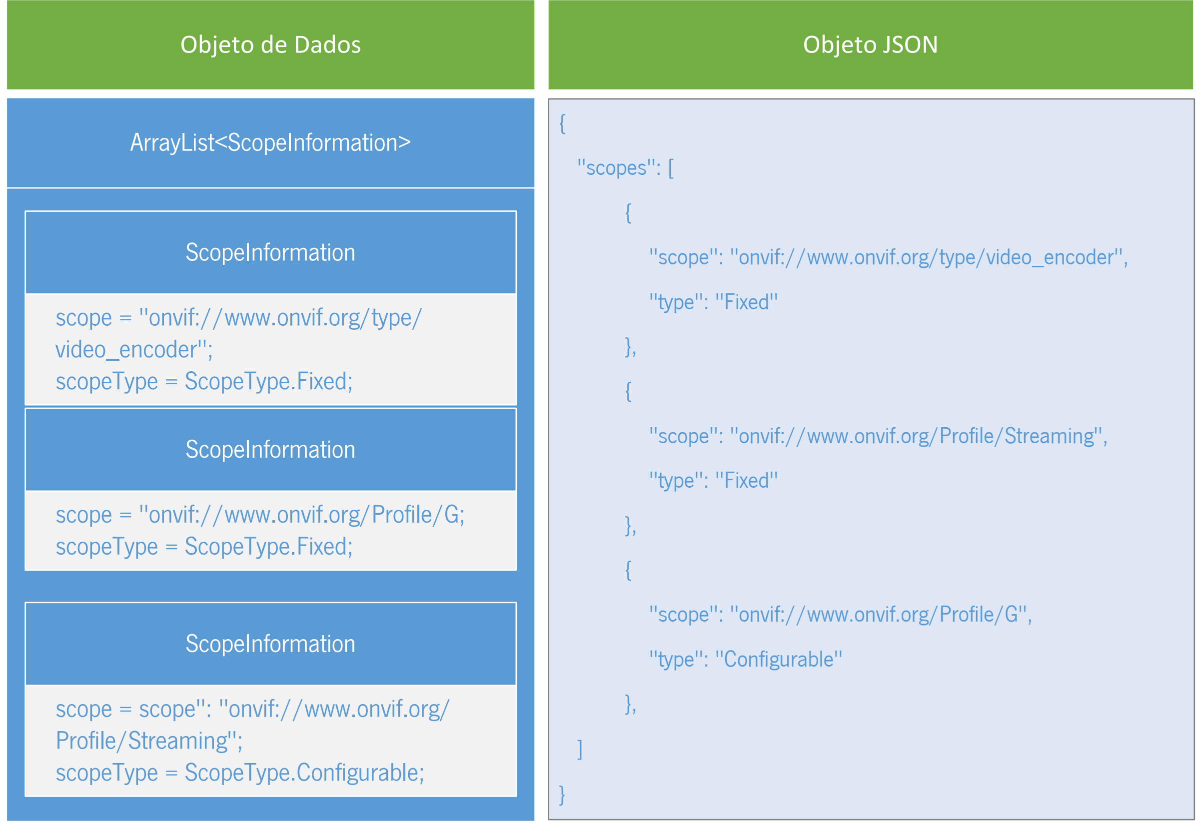


Figura 4.10 - Equivalência entre um objeto dados e um objeto JSON para uma coleção de *scopes*

A biblioteca JSON de Java pode gerar *exceptions,* que são *checked*. Na análise desta *exception* concluiu-se que ocorre devido a erros de implementação no servidor ou na biblioteca, pelo que, não

devem acontecer. De modo a evitar que a aplicação lide com estas *exceptions* (simplificando o uso da biblioteca e implementação da aplicação) criou-se um *exception* BadJSONData derivada de RuntimeException [50]. Esta é apanhada e relançada como *unchecked*.

4.3.4 Classes de suporte a métodos assíncronos

As AsyncTasks permitem executar operações assíncronas a partir da UI Thread e fornecem *handlers* para lidar com o progresso, erros e resultados. Além disso, os *handlers* têm a capacidade de fazer alterações na UI diretamente. Os métodos assíncronos utilizam a AsyncTask, que por sua vez invoca diretamente um método síncrono. Para métodos síncronos sem retorno e com retorno criaram-se respetivamente as classes OnvifAsyncTask e OnvifAsyncTaskNoResult. A classe OnvifAsyncTask tem *listeners* para lidar com o resultado, progresso e erros enquanto a OnvifAsyncTaskNoResult tem *listeners* para lidar apenas com os erros e resultado. No diagrama de classes da Figura 4.11 pode-se consultar os campos e métodos para classes derivadas da AsyncTask.

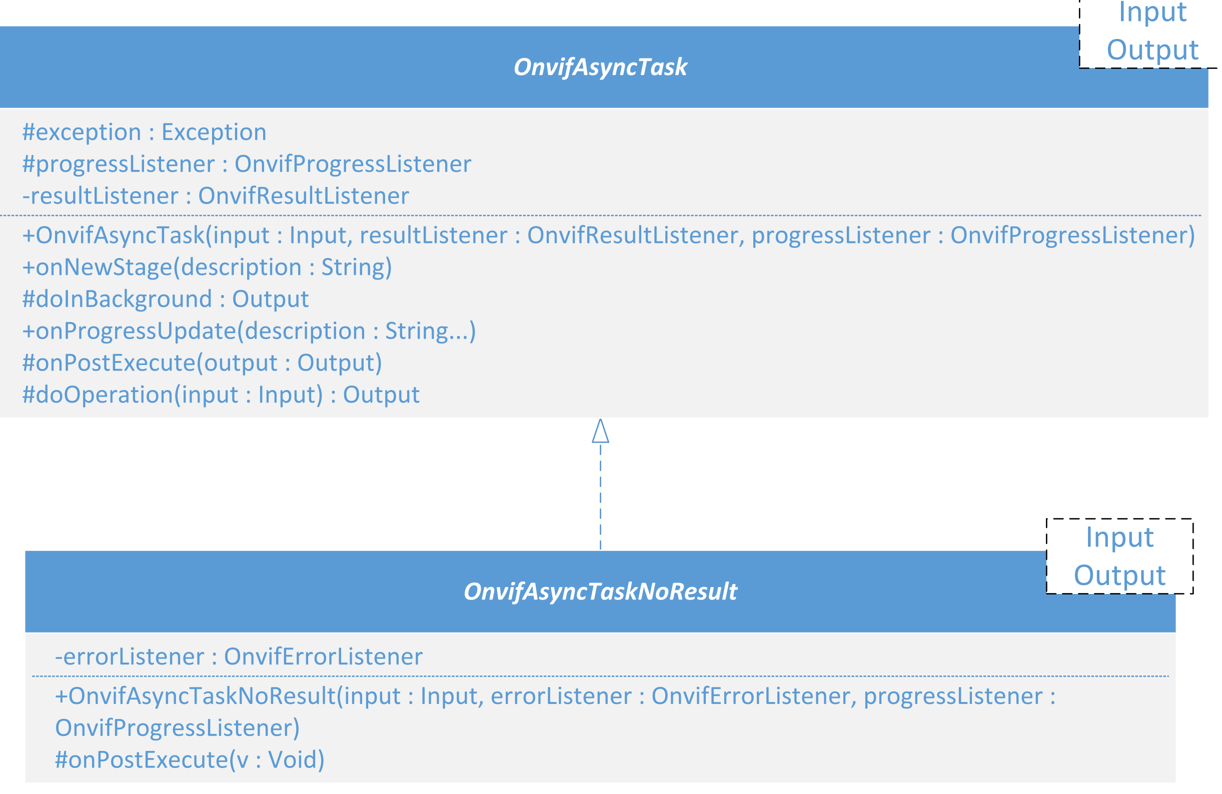


Figura 4.11 - Diagrama das classes OnvifAsyncTask e OnvifAsyncTaskNoResult

As AsyncTasks implementam o OnvifProgressListener de modo a poderem ser argumentos de entrada dos métodos síncronos e atualizarem o progresso como no caso da utilização de um objeto

OnvifProgressListener. A razão desta escolha é explicada no capitulo 4.2.2 - Classes de serviço que refere quais as decisões de desenho tomadas para implementação do progresso.

A implementação da OnvifAsyncTask é feita da seguinte forma:

- Tem um campo do tipo Exception para guardar as exceções ocorridas no doInBackground();
- No doInBackground() da AsyncTask vai-se fazendo publishProgress() dos passos intermédios e invoca-se a versão síncrona referente ao método assíncrono que se pretende realizar. Os métodos síncronos recebem a AsyncTask de modo a atualizarem o progresso de acordo com os passos intermédios. Adicionalmente invoca-se o *handler* onError() passando o erro ocorrido para a UI;
- No onProgressUpdate() da AsyncTask invoca-se o *handler* onProgress() para ser executado na UI Thread e poder ser utilizado para fazer atualizações;
- No onPostExecute() da AsyncTask, invoca-se o *handler* onResult().

A OnvifAsyncTaskNoResult não tem resultado e consequentemente não invoca o *handler* onResult() não necessitando do método herdado da AsyncTask onPostExecute(). Para lidar com o onResult(), implementa-se o método onPostExecute() na classe OnvifAsyncTask que deriva da OnvifAsyncTaskNoResult, herdando todos os métodos já implementados.

Assim, as AsyncTaks oferecem as funcionalidades necessárias para receber dados, devolver dados, invocar exceções e atualizar o progresso.

4.3.5 Classes de dados

No modelo de dados as instanciações dos objetos são feitas através do construtor. Os campos dos construtores podem ser objetos de dados e tipos primitivos de Java que correspondem aos objetos JSON do serviço REST. Os dados que são opcionais, são um caso particular, pois podem não estar no conteúdo de uma resposta e por isso coloca-se o valor *null*. Por exemplo, a classe de dados DateTimeInformation recebe dois parâmetros opcionais, "local" e "utc". Quando se obtém a data e a hora pode-se receber na resposta apenas os dados do objeto "local". Neste caso um objeto de dados do tipo DateTimeInformation é instanciado com a expressão "new DateTimeInformation(local, null, ...)" em que o objeto "utc" fica com o valor *null*.

Os diagramas de classes de dados para todos os serviços e respetiva descrição podem ser consultados no Anexo V – Classes de Dados.

4.3.6 Classe OnvifCamera

As classes de serviço providenciam métodos que abstraem as funções ONVIF. Os objetos destas classes são disponibilizados pela OnvifCamera. A classe OnvifCamera é a classe principal e tem campos para o URI, *username, password* e um *singleton* (OnvifRestServer) para guardar e obter o endereços REST. Invoca no construtor um método privado getServices para obter os serviços suportados que usa um pedido no formato GET /onvif/devicemgmt,{ESC}/Services. Este método é público na classe de serviço DeviceMangment para permitir ao programador instanciar classes de serviço sem recorrer à classe OnvifCamera. Os serviços são obtidos de forma síncrona e é feita a instanciação dos objetos de serviço suportados. Por sua vez cada objeto de serviço obtém as suas *capabilities* também de forma síncrona. O diagrama de classes com a relação entre a classe OnvifCamera e as classes de serviço pode ser consultado na Figura 4.12 onde se verifica que os objetos de serviço podem ser nulos caso sejam opcionais. Os objetos de serviço obtêm pelo construtor as credenciais e o endereço do servidor da OnvifCamera uma vez que efetuam pedidos que necessitam desses dados.

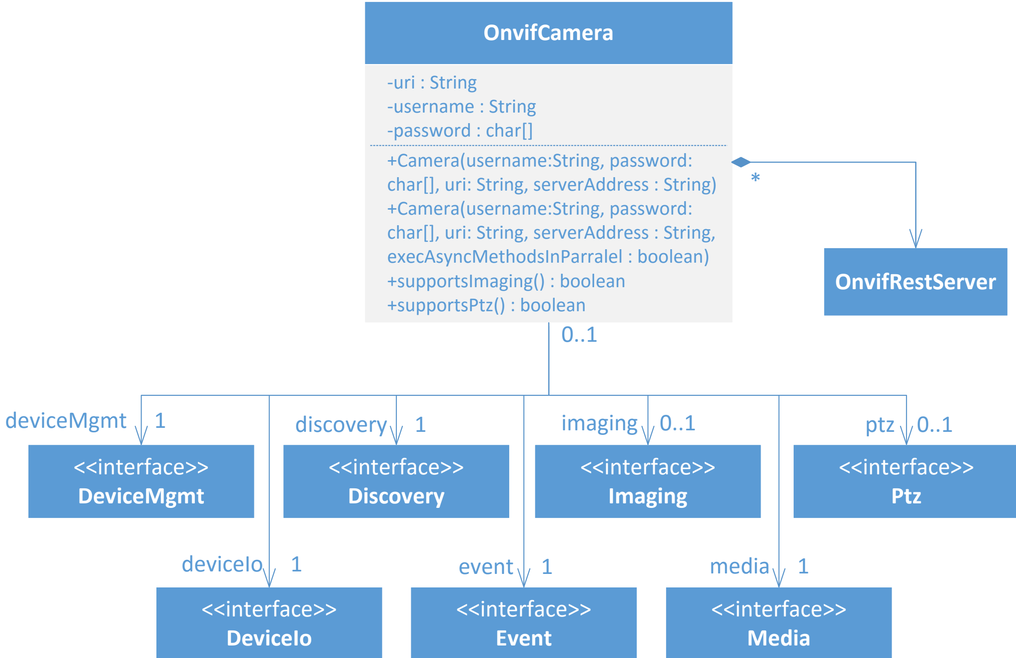


Figura 4.12 - Diagrama de classes para a OnvifCamera

O fator mais importante deste desenho é que com a inicialização da classe de serviço através da interface pode-se recorrer ao construtor da classe <Serviço>Rest ou <Serviço>JNI de acordo com o modo de comunicação que se pretende utilizar. As interfaces de serviço JNI são de um trabalho paralelo podendo ser facilmente adicionadas à API sem alteração da mesma. A OnvifCamera tem dois

construtores um com entrada do parâmetro do endereço REST e outro sem esse parâmetro. Os construtores da classe OnvifCamera conhecem os dois modos de comunicação, ou seja, conhecem os construtores das classes de serviço. A partir disto as variáveis dos objetos de serviço são instanciadas de acordo com o modo de comunicação, as quais, estão disponíveis independentemente do modo de comunicação. Em conclusão, a instanciação das interfaces de serviço permite abstrair o uso do modo de comunicação JNI, REST ou novos modos. A classe que implementa os métodos assíncronos não depende do tipo de arquitetura não sendo necessário classes adicionais.

4.3.7 Classes de serviço

Existem relações entre as classes de serviço, *listeners e* AsyncTaks, que podem ser consultadas na Figura 4.13, a qual, usa só as classes do serviço PTZ como exemplo. O diagrama para os outros serviços é semelhante. O construtor da classe de serviço faz o pedido GET /onvif/<serviço>,{ESC}/Capabilities para saber quais as funcionalidades suportadas. Nesta classe, as *capabilities* obtidas pelo GetServiceCapabilities são a primeira operação de rede que se realiza. O endereço de serviço codificado é obtido pelo getServices que tem o parâmetro “xAdr” para utilização na URI dos serviços, sendo que, a substring “http://” é filtrada na construção do URI utilizado na comunicação. As *capabilities* e serviços suportados são essenciais para o funcionamento da biblioteca.

O diagrama de classes apresenta 3 níveis para as classes de serviço. Os métodos da biblioteca são definidos na interface de serviço no primeiro nível. No segundo nível tem-se uma classe abstrata que implementa os métodos assíncronos. Os métodos síncronos são feitos no terceiro nível onde efetivamente se realizam as operações de rede para um modo de comunicação. Este nível é o local onde se adiciona classes para outros modos de comunicação que se queiram implementar.

Os métodos dos serviços fazem uso de um *listener* para atualizar o progresso. Decidiu-se criar o OnvifNullProgressListener para quando o programador não quer informações de progresso. Esta classe implementa o padrão de desenho Null Object [51], ou seja, as invocações deste *listener* são vazias, o que simplifica a implementação e reduz as hipóteses de erro, relativamente a fazer comparações para determinar se é ou não necessário fornecer informações de progresso. Como há métodos que não têm retorno é utilizado o OnvifErrorListener para fornecer informações de erro, e no caso de métodos com retorno é utilizado o OnvifResultListener que fornece informações de erro e de resultado.

Os métodos síncronos utilizam apenas um objeto OnvifProgressListener como argumento de entrada para atualizar o progresso. Não necessitam de mais nenhum *listener* porque o resultado é um parâmetro de retorno e o tratamento dos erros é feito por lançamento de exceções. Os argumentos de entrada dos métodos assíncronos são um objeto OnvifResultListener para lidar com o resultado/erros e OnvidProgressListener para atualizar o progresso.

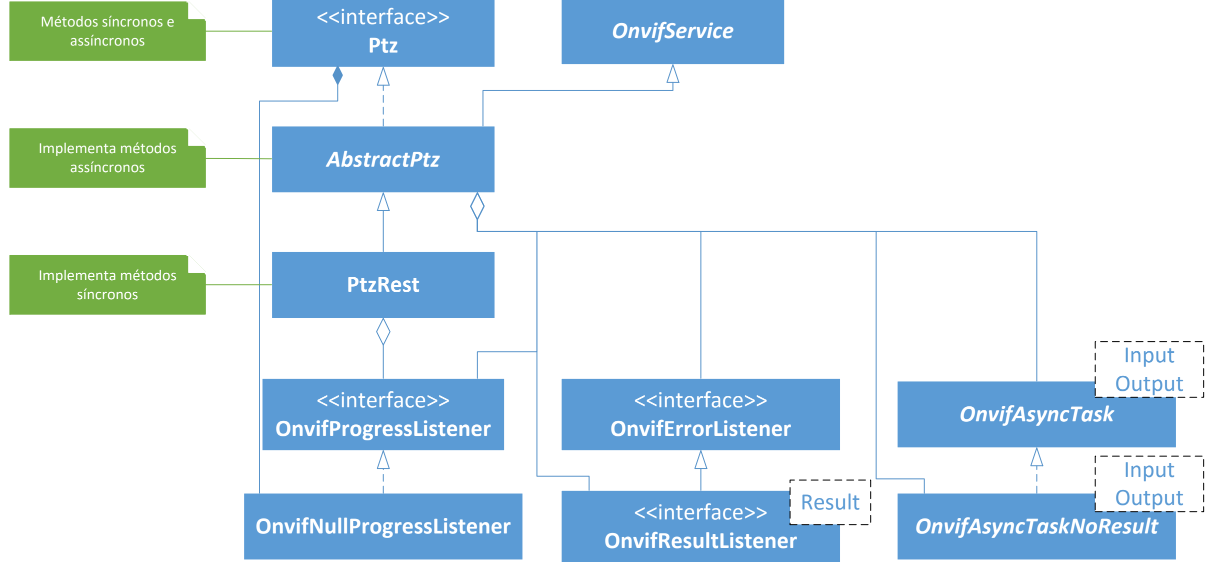


Figura 4.13 - Diagrama de classes para a relação dos serviços com os *listeners* e AsyncTakss

Foi criada a classe abstrata OnvifService (Figura 4.14), que é herdada pelas classes de serviço, pois todas elas partilham os parâmetros para as credenciais da câmara, o tipo de Executor e o tipo do

listener de progresso. Os construtores desta classe fazem essas inicializações e são usados pelos construtores das classes de serviço.

OnvifService

#username : String
#password : String
#uri : String
#asyncTaskExecutor : Executor
#nullProgressListener : OnvifNullProgressListener = new NullProgressListener()

+OnvifService(username : String, password : char[], uri : String)
+OnvifService(username : String, password : char[], uri : String, execAsyncMethodsInParralel : boolean)
#checkProgressListener(progressListener : OnvifProgressListener) : OnvifProgressListener

Figura 4.14 - Diagrama da classe abstrata OnvifService

A título de exemplo, o diagrama de classes para o serviço Device IO é exibido na Figura 4.15 onde podem ser consultados os construtores e métodos implementados. A classe AbstractDeviceIO e DeviceIoRest têm uma linha com "..." no local dos métodos pois os mesmos correspondem respetivamente aos métodos assíncronos e síncronos da interface DeviceIO. Esta lógica é aplicada nos restantes diagramas de classes de serviço para facilitar a leitura e visualização dos mesmos devido à quantidade considerável de métodos existentes.

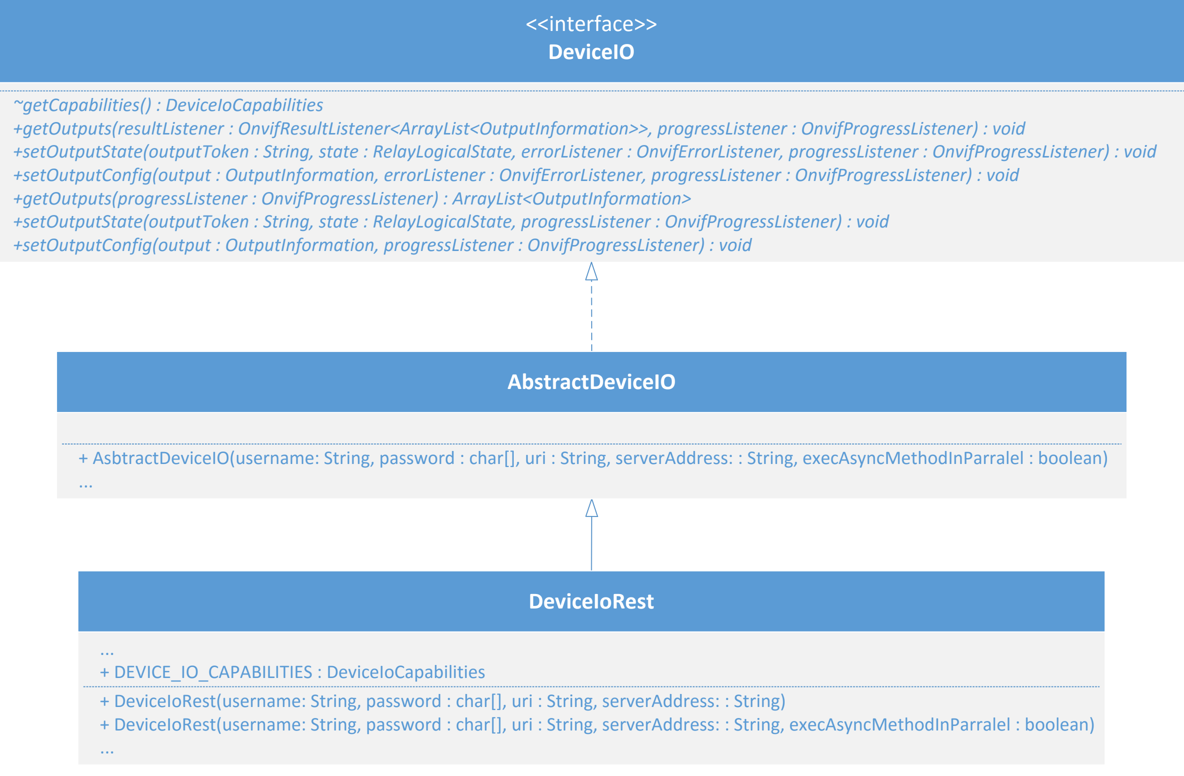


Figura 4.15 - Diagrama de classes de serviço do Device IO

Os diagramas de classes e respetiva descrição para os restantes serviços podem ser consultados no Anexo VI – Classes de Serviço.

4.3.8 Métodos síncronos

Os métodos síncronos realizam tarefas diretas de rede. Estes utilizam apenas um objeto OnvifProgressListener como argumento de entrada para atualizar o progresso. Não necessitam de mais nenhum *listener* porque o resultado é um parâmetro de retorno e o tratamento dos erros é feito por lançamento de exceções. Por exemplo, a declaração de um método síncrono para obter o Outputs do Device IO tem o formato "ArrayList<OutputInformation> getOutputs(OnvifProgressListener)".

A serialização e desserialização dos objetos JSON é feita a nível destes métodos, sendo o único local da API onde isto é feito. O conteúdo JSON recebido pelos métodos é obtido a nível de classes de comunicação que devem estabelecer a conexão e obter o conteúdo JSON. Quando um método pretende enviar/receber informação JSON as classes de comunicação são responsáveis por estabelecer a conexão e enviar/receber o conteúdo JSON nas mensagens trocadas.

4.3.9 Métodos assíncronos

Os métodos assíncronos executam operações de rede em *background,* invocam o método síncrono para obter resultados e conseguem publica-los diretamente na UI Thread. Estes métodos, utilizam duas variações das AsyncTaks, com e sem retorno de dados. A estrutura para fazer as variantes dos métodos é semelhante sendo que as AsyncTaks criadas necessitam apenas de parâmetros de entrada sobre o método síncrono que se pretende invocar. Para isso foram criados dois *templantes* que facilitam a criação de métodos assíncronos: Template de métodos assíncronos com retorno (Figura 4.16) e de métodos assíncronos sem retorno (Figura 4.17). Os argumentos de entrada do tipo *listener* dos métodos assíncronos são um objeto OnvidProgressListener para atualizar o progresso e o OnvifResultListener/OnvifErrorListener para lidar com o resultado e erros ou só com os erros quando não há resultado.

Os valores a de cor preta dos *templates* da Figura 4.16 e da Figura 4.17 devem ser substituídos pelos campos pretendidos como o nome, tipo e parâmetros do método. Um exemplo da aplicação do *template* (com retorno) para fazer o método assíncrono getSnapshotURI é ilustrado na Figura 4.18. A implementação é simples através da aplicação do *template*. As futuras adições à

biblioteca destes métodos e a implementação atual ficam assim simplificadas pela estrutura dos *templates*. Os parâmetros dos métodos síncronos são naturalmente recebidos pelos assíncronos.

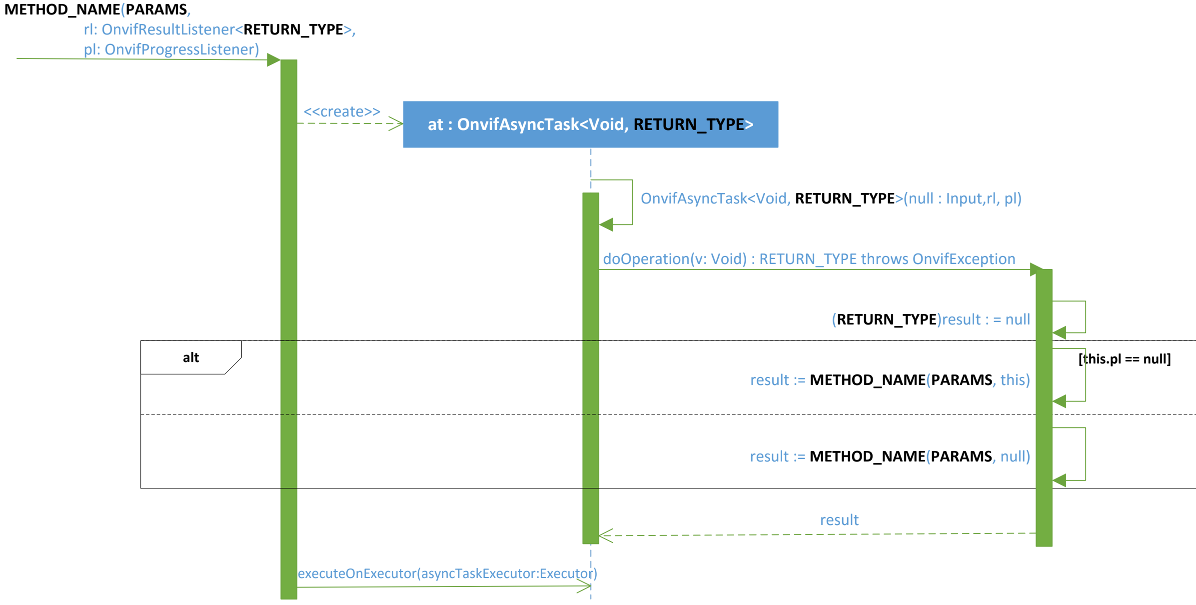


Figura 4.16 - Diagrama sequência do *template* dos métodos assíncronos com retorno

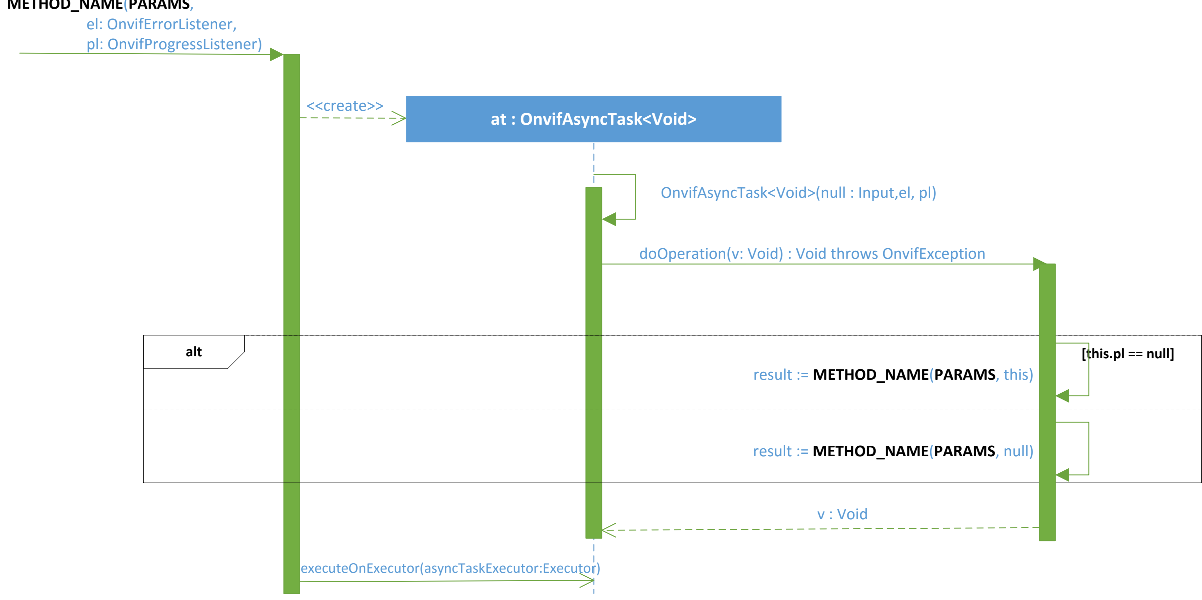


Figura 4.17 - Diagrama sequência do *template* dos métodos assíncronos sem retorno

```
@Override
public void getSnapshotURI(
        String mediaProfile,
        OnvifResultListener<String> resultListener,
        OnvifProgressListener progressListener) {
    final OnvifAsyncTask at = new OnvifAsyncTask<Void, String>
            (null/*Input input*/, resultListener, progressListener) {
        public String doOperation(Void v) throws OnvifException {
```

```
            String result = null;
            if (this.progressListener!=null)
                result = getSnapshotURI(mediaProfile, this);
            else
                result = getSnapshotURI(mediaProfile, null);
            return result;
        }
    };
    at.executeOnExecutor(asyncTaskExecutor);
}
```

Figura 4.18 – Implementação do método getSnapshotURI, versão assíncrona.

4.3.10 Informação de Progresso

O fornecimento de informação de progresso é uma funcionalidade que pode implicar mudanças no desenho inicial e descrito até ao momento e que influência desempenho. De modo a desenhar a funcionalidade, fez-se uma análise temporal de diversas fases dos métodos em que se pode fornecer informação de progresso. As medições de tempo pretendem avaliar qual o tipo de progresso que se pretende fornecer, nomeadamente qualitativo ou quantitativo. Além disso, pretende-se que o programador decida se necessita da informação de progresso, ou seja, as informações de progresso são opcionais. No caso de o programador prescindir da informação de progresso, a solução deve evitar processamento desnecessário e/ou extra, para otimizar o desempenho.

Os testes foram feitos num dispositivo real, com conexão à rede estável e através do servidor REST alojado na Universidade do Minho. Consideram-se estas condições normais e adequadas na utilização da biblioteca porque proporcionam a quantidade adequada de saltos de rede e conetividade. Não se considerou a largura de banda da rede disponível porque interessa apenas comparar proporções de tempos numa situação real, sendo que, estas serão semelhantes independentemente da largura de banda. Os tempos de processamento de informação não são comparáveis com tempos que dependem da rede, sendo que, é previsível que os tempos de rede sejam bastante maiores que tempos de processamento.

Mediram-se os intervalos de tempo através da classe System e da função estática nanoTime(). Escolheu-se esta função porque retorna o valor atual do temporizador do sistema mais preciso, em nanossegundos [52]. Converteram-se as unidades para milissegundos porque os algarismos significativos estavam nessa ordem, facilitando assim, a leitura e análise dos valores.

Existem 6 tipos possíveis de tempos em análise que dependendo do tipo de pedido podem ser distintos e até nulos. O tempo de serialização existe em pedidos que têm dados para enviar no pedido que precisam ser serializados, sendo que, pode ser nulo para pedidos do tipo GET e DELETE. O tempo de conexão existe sempre e representa o tempo em que se inicia a conexão e se prepara o cabeçalho

do pedido. No envio de informação é calculado o tempo que a informação serializada demora a ser transmitida para o servidor, sendo que, é nulo quando o tempo de serialização é nulo. Na receção de informação é medido o tempo que demora a receber a informação da resposta do servidor para posterior desserialização. Por fim, é medido o tempo de desserialização da resposta recebida.

O tempo de rede é obtido através da soma do tempo de conexão com os de receção e envio. O tempo total é a soma do tempo de rede com os tempos de serialização e desserialização. A proporção do tempo de rede é obtida através da razão entre as médias do tempo de rede e do tempo total multiplicado por 100. A fórmula usada para calcular a proporção de rede é a seguinte:

$$Proporção\ do\ tempo\ Rede = \frac{\overline{tempo\ de\ rede}}{\overline{tempo\ total}} * 100$$

Para os pedidos em análise, os tempos podem variar dependendo das condições de rede e do dispositivo físico, sendo que, fizeram-se 30 pedidos de teste e recolheram-se os 10 pedidos com menor desvio. Esta escolha pretende que se analise uma situação comum e como base de apoio ao estudo da proporção do tempo de rede não sendo necessário efetuar uma quantidade volumosa de pedidos.

O método GetMediaProfiles é dos pedidos que pode conter maior quantidade de informação dependendo dos perfis existentes na câmara que contribuem para a variação do tamanho da resposta. Não há tempo de serialização e consequentemente não há tempo de envio de dados para este pedido.

A proporção do tempo de rede obtida para o pedido GetMediaProfiles é 93.36%. A Figura 4.19 contém o gráfico de barras do pedido GetMediaProfiles das médias temporais em milissegundos. O tempo de receção é o que se destaca relativamente aos outros uma vez que este pedido pode devolver informações sobre vários perfis de media configurados no dispositivo. Este é um dos fatores para a proporção do tempo de rede ser elevada.

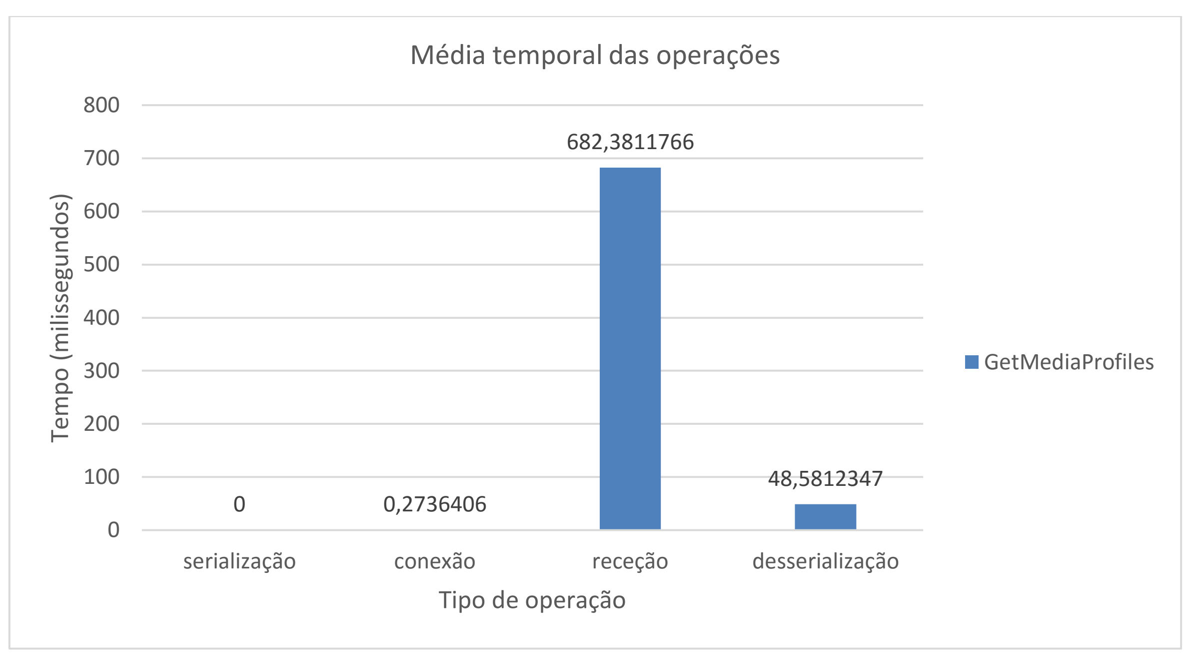


Figura 4.19 - Médias temporais do pedido GetMediaProfiles

Naturalmente, o tempo de receção é o mais elevado porque é necessário receber as informações de todos os perfis de media através da rede. O tempo de conexão é mínimo, pois é apenas iniciada a conexão e definido o tipo de pedido através do preenchimento do HTTP Header. O tempo de desserialização é significativamente menor que o tempo de receção pois não é influenciado pela rede e converte os dados recebidos em objetos das classes de dados.

Utilizou-se também o SetSystemDateAndTime, pois envia dados (em vez de receber) e tem um tamanho das mensagens previsível. Neste existem dados para serializar e consequentemente existe tempo de envio de informação que vai no corpo do pedido, pelo que, estes tempos não serão nulos.

A proporção do tempo de rede obtida para o pedido SetSystemDateAndTime é 99.97%. A Figura 4.20 exibe o gráfico de barras do pedido SetSystemDateAndTime das médias temporais em milissegundos. Devido ao tempo de receção a proporção do tempo de rede é quase 100%.

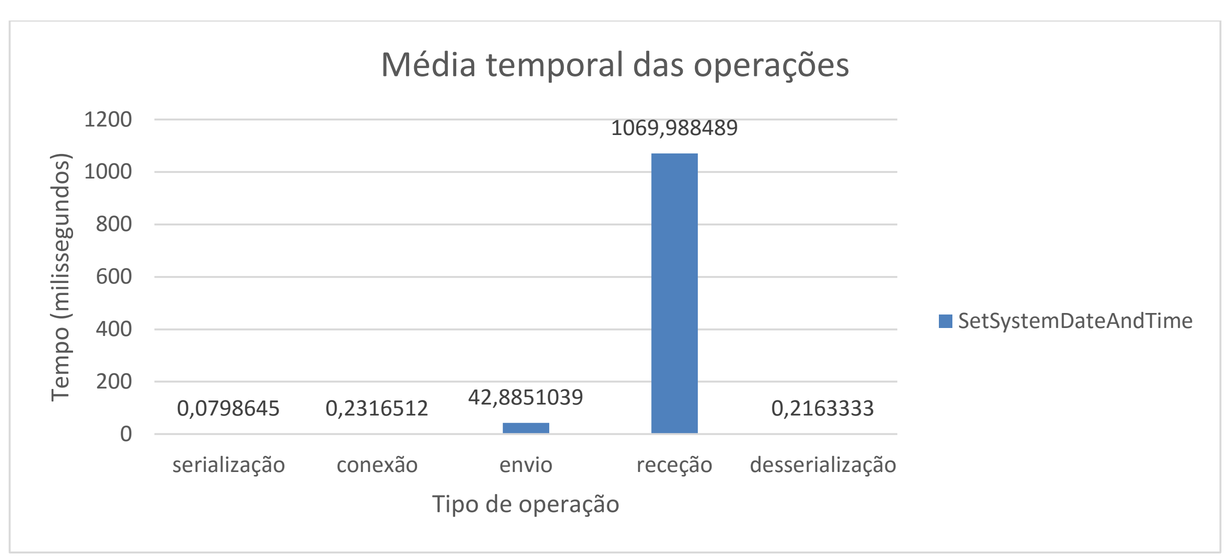


Figura 4.20 - Médias temporais do pedido SetSystemDateAndtime

Tal como no pedido GetMediaProfiles, o SetSystemDateAndTime tem um tempo de receção elevado devido a troca de dados através da rede. Neste caso existe tempo de envio da informação que foi serializada e como a informação recebida é menor, o tempo de desserialização é menor.

O tempo de rede em relação ao tempo total dos métodos permitem verificar se o progresso quantitativo é relevante. Como a média dos tempos de rede em relação ao tempo total são 93.36% e 99.97% significa que o tempo de rede é crítico na medida do progresso. As velocidades de transferência de dados variam, assim como, o tamanho das mensagens o que não permite a medição correta do progresso quantitativo durante o tempo de rede. Como a proporção do tempo de rede é muito elevada, conclui-se que, o progresso quantitativo não é adequado. Em conclusão escolhe-se o progresso qualitativo que pode fornecer informações mais significativas sobre o estado das operações.

Escolheu-se a solução que disponibiliza progresso nos métodos síncronos e assíncronos através de mensagens no formato de *strings*. A quantidade e descrição das mensagens de progresso variam de método para método de acordo com as operações de cada um. O diagrama de sequência da Figura 4.21 corresponde ao desenho final da solução com progresso. Este mostra os *listeners* de progresso e resultado que são utilizados nas operações, sendo que a versão síncrona só utiliza o segundo, uma vez que o resultado é obtido por retorno. A operação assíncrona cria uma AsyncTask, passando-lhe como argumentos o objeto serviço e o *listener* de progresso, e a seguir lança a execução da AsyncTask (executeOnExecutor). A AsyncTask invoca o método síncrono passando o valor *null* se o *listener* de progresso estiver a *null,* o que faz, com que as atualizações de progresso usem o objeto nulo de progresso para fazer invocações vazias quando é invocado o método onNewStage(). Caso o progresso não seja nulo a AsyncTask (que implementa o OnvifProgressListener) é passada na

invocação do método síncrono para atualizar o progresso. Quando o método síncrono envia o progresso para o *listener* AsyncTask (onNewStage), esta, invoca o método publishProgress() da AsyncTask e usa o onProgressUpdate() para publicar o progresso no *listener* recebido no método assíncrono. Desta forma usa-se as *callbacks* da AsyncTask que permitem trabalhar diretamente na UI Thread. A solução do progresso para a versão síncrona e assíncrona fica assim explicita num só diagrama que conjuga a solução para os dois tipos de métodos possíveis.

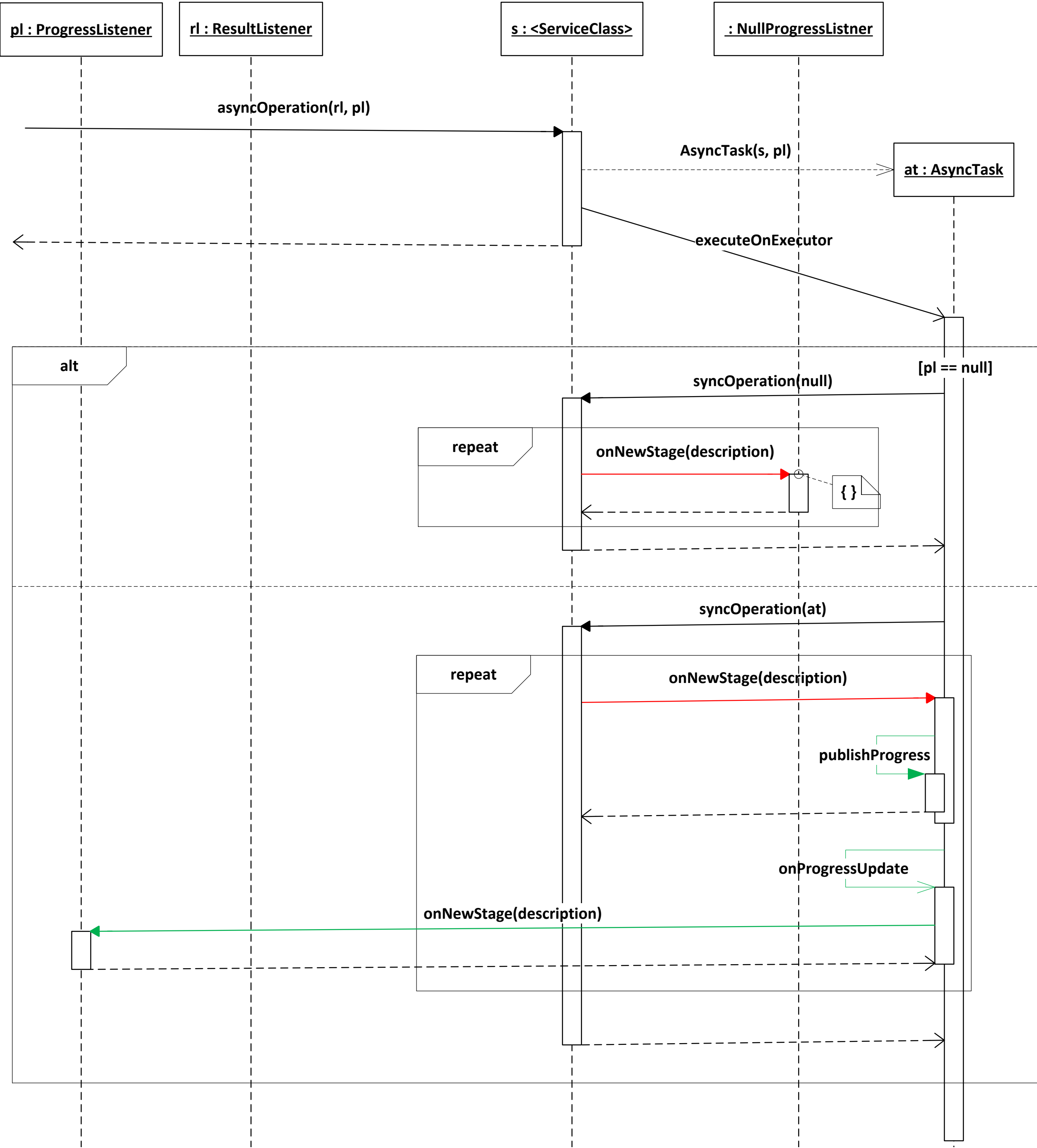


Figura 4.21 - Diagrama de sequência para a solução final do progresso da biblioteca

O desenho da solução de progresso final é a terceira versão de uma série que foi sendo aperfeiçoada. A primeira versão fornecia apenas progresso nos métodos assíncronos, sendo que nas versões seguintes se introduziu progresso nos métodos síncronos e fez-se melhoramentos nas AsyncTasks criadas e nos *handlers* da biblioteca. Os diagramas de sequência projetados foram um instrumento ideal e decisivo no desenho da versão final, permitindo focar em simultâneo as interações entre objetos e a sequência das operações.

Em conclusão da análise dos resultados e do desenho optado registou-se vantagens e contras da utilização desta solução:

As vantagens são as seguintes:

- Permite fornecer informação qualitativa de progresso: serializar Java para JSON, estabelecer a conexão, preparar o cabeçalho, enviar dados, receber dados, desligar a conexão, e desserializar JSON para Java.
- Pode ser ignorada fornecendo um *listener null*
- O impacto sobre o desempenho é marginal.

Os contras são os seguintes:

- Que benefícios pode o programador da aplicação retirar da informação fornecida? No caso de não ser utilizado a informação do progresso é processamento extra, embora, minimizado no desenho.

Numa aplicação a indicação de que a operação está a ser realizada em *background* pode ser dada utilizando uma ProgressBar em modo indeterminado. Se for escolhida a ProgressDialog a UI é bloqueada para apresentar a informações. Para os métodos assíncronos pode não interessar bloquear a UI enquanto para os síncronos interessa.

4.3.11 Listeners disponibilizados

Criaram-se quatro classes *listeners* para utilização na biblioteca que são disponibilizados ao programador. Cada um dos *listeners* tem apenas um método descrito no diagrama de classes da Figura 4.22. O OnvifResultListener herda o método do OnvifErrorListener de modo a se utilizar um único *listener* quando há resultado.

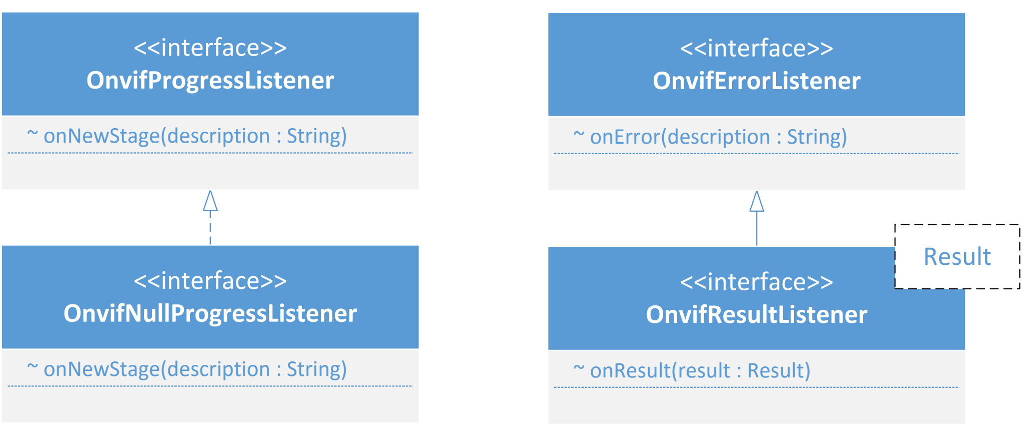


Figura 4.22 - Diagrama de classes dos listeners

A classe OnvifProgressListener herda a classe OnvifNullProgressListener porque a atualização de progresso é opcional, pelo que, a nível de desempenho é preferível ter um objeto que invoque um método vazio invés de recorrer a condições para atualizar o progresso. Esta particularidade permite evitar no código a comparação de se um objeto é nulo ou se é instância da classe OnvifProgressListener sempre que se quer atualizar o progresso. O Result do OnvifResultListener é um tipo genérico para que o resultado possa ser do tipo das classes de dados.

4.4 Utilização da biblioteca

Há diversas formas de uso da biblioteca devido ao desenho projetado, sendo que, neste subcapítulo são exemplificadas algumas formas de uso da mesma. Os exemplos servem de referência para o modo de utilização da biblioteca. As classes usadas estão documentadas obedecendo às regras de documentação de classes Javadoc [53]. Esta documentação tem a descrição dos métodos, classes de dados, classes de serviço e construtores. Além disso a documentação pode ser consultada via *web* oferecendo uma navegação facilitada.

Para usar a biblioteca o programador tem a opção de criar um objeto OnvifCamera que automaticamente disponibiliza os objetos dos serviços disponíveis, ou cria um objeto de serviço DeviceMgmt, invoca o getServices e cria os objetos de serviço que necessitar (entre os disponíveis). A primeira opção é mais prática, mas a segunda é mais eficiente quando se necessita de apenas alguns dos serviços. A partir dos objetos de serviço é possível invocar os métodos pretendidos nas versões síncronas e assíncronas. O exemplo de criação de um objeto da classe OnvifCamera pode ser consultado na Figura 4.23. Os construtores de cada serviço têm os mesmos argumentos de entrada que a classe OnvifCamera, mudando o endereço de serviço, que é obtido no getServices(). Um

exemplo de acesso a um objeto de serviço da classe OnvifCamera pode ser consultado na Figura 4.24. Um exemplo da criação de objetos de serviço pode ser consultado na Figura 4.26. Para verificar se o serviço é suportado usa-se os métodos da OnvifCamera. Os serviços suportados podem ser verificados por métodos como no exemplo da Figura 4.25 para o serviço PTZ.

Na comunicação HTTPS, o certificado existente é assinado pelo próprio servidor em alternativa ao uso de um certificado de uma autoridade certificada para as assinaturas. Deste modo, quando se estabelece uma conexão, o certificado não é confiável, o que requer a criação dos métodos "trustAllHosts()" para confiar em todos (pouco seguro) e "trustHost(String hostname, Integer cRawCrt)" para confiar num certificado especifico (preferível). Caso o programador opte por não utilizar a classe de utilidades tem de confiar no certificado ou utilizar um certificado assinado por uma autoridade.

Nos exemplos de utilização os valores dos campos são apenas ilustrativos.

```
final user = "username";
final char[] password = new char[]{'p', 'a', 's', 's', 'w', 'o', 'r','d'};
String deviceService = "193.123.123.123";//device address
String restWS = "https://193.123.123.124";//server address
boolean execAsyncMethodsInParralel = true;
OnvifCamera camera = new OnvifCamera(user,
                                     password,
                                     deviceService,
                                     restWS,
                                     execAsyncMethodsInParralel
                                     );
```

Figura 4.23 - Criação de um objeto da OnvifCamera

```
final DeviceMgmtRest deviceMgmt = camera.deviceMgmt;
```

Figura 4.24 - Acesso a um objeto de serviço da OnvifCamera

```
//returns true if the camera supports PTZ
if(camera.supportsPTZ())
     camera.ptz.goToHomePosition("profile", progressListener);
```

Figura 4.25 - Verificar se serviço é suportado pela câmara

```
final DeviceMgmtRest deviceMgmtRest =
                new DeviceMgmtRest(
                                   user,
                                   password,
                                   deviceService,
                                   restWS,
                                   execAsyncMethodsInParralel
                                   );
NVTServices services = deviceMgmt.getServices(null);
if(services.imaging!=null){
    String imagingService = removeHtppOrHttps(services.imaging.uri);
    imagingService = URLEncoder.encode(imagingService,"UTF-8");
    imaging = new Imaging(
                        user,
                        password,
                        imagingService,
                        restWS,
                        execAsyncMethodsInParralel
                        );
    }
```

Figura 4.26 - Criação dos objetos de serviço Device Managment e Imaging

Para criar um objeto OnvifErrorListener usa-se o exemplo da Figura 4.27. A criação do OnvifProgressListener é semelhante, muda o nome do *listener*, e o nome do método para "onNewStage(String description)". Para o OnvifResultListener usa-se os métodos onResult(Result result) e onError(String description).

```
OnvifErrorListener errorListener = new OnvifErrorListener() {
    @Override
    public void onError(String description) {
        //code to handle errors
    }
};
```

Figura 4.27 - Criação de um objeto OnvifErrorListener

Para invocar um método sincrono com retorno utiliza-se o exemplo da Figura 4.28.

```
ArrayList<AccountInformation> accounts;
try {
    accounts = deviceMgmtRest.getUsers(progressListener);
    } catch (OnvifException e) {
    //code to handle OnvifException
    }
}
```

Figura 4.28 - Invocação de um método síncrono com retorno

Para invocar um método síncrono sem retorno utiliza-se o exemplo da Figura 4.29.

```
try {
    deviceMgmtRest.deleteUsers("user", progressListener);
} catch (OnvifException e) {
    //code to handle OnvifException
}
```

Figura 4.29 - Invocação de um método síncrono sem retorno

Um exemplo de invocação de um método assíncrono com retorno pode ser consultado na Figura 4.30.

```
deviceMgmtRest.getUsers(resultListener, progressListener);
```

Figura 4.30 - Invocação de um método assíncrono com retorno

Um exemplo de invocação de um método assíncrono sem retorno é exibido na Figura 4.31.

```
deviceMgmtRest.deleteUsers("user", errorListener, progressListener);
```

Figura 4.31 - Invocação de um método assíncrono sem retorno

Na invocação dos métodos caso não se pretenda obter o progresso substitui-se o parâmetro do progressListener por "null".

5. APLICAÇÃO ANDROID

Este capítulo explica o novo desenho da aplicação assim como tem um resumo da estrutura e funcionalidades das aplicações existentes. Inicialmente é feito um levantamento de requisitos baseado no conceito das aplicações anteriores de modo a acrescentar melhorias e novas funcionalidades à nova aplicação. Pretende-se assim, obter uma aplicação mais completa e melhorada.

5.1 Requisitos

Fez-se o levantamento de requisitos funcionais e não funcionais. Os requisitos são importantes para definir os objetivos pretendidos para a aplicação, podendo assim tomar opções de desenho adequadas e otimizar a implementação do mesmo.

Requisitos funcionais

- Utilizar a biblioteca Java para comunicar com as câmaras através do serviço REST.
- Gerir várias câmaras ONVIF.
- Suportar as operações do serviço web REST.
- Exibir o *stream* de vídeo de uma câmara.
- Suportar comunicação segura.

Requisitos não funcionais

- Reduzir o número de pedidos realizados ao serviço REST.
- Fornecer uma interface com o utilizador consistente entre os vários serviços.
- Ter preocupações com o desempenho.
- Otimização de consumo de bateria do dispositivo móvel.

5.2 Aplicações de base revisitadas

Nas subsecções deste capitulo é feita a descrição das funcionalidades, estrutura da GUI e comportamento das aplicações. A arquitetura das aplicações de base foi descrita no capitulo 2.3.5 Aplicações base.

5.2.1 Aplicação com suporte para alguns serviços

A nível estrutural a aplicação para serviços contém fragmentos para os serviços Device Managment, Device IO, Media e Imaging, e uma atividade de *streaming* para o serviço PTZ. Em adição utiliza uma atividade inicial para gestão de câmaras e uma atividade secundária que dá acesso aos fragmentos de serviço com recurso à barra de navegação lateral.

As funcionalidades são:

- É possível adicionar até um total de 5 câmaras no menu inicial onde é divulgado o estado da ligação à câmara. Em adição é possível editar ou eliminar os dados de cada câmara configurada.
- No Device Managment é possível gerir utilizadores de câmaras, scopes, data e hora e configurar definições de rede. Neste serviço é possível consultar as capabilities de todos os serviços.
- No serviço de Device IO é possível gerir as saídas existentes das câmaras.
- No serviço de Media são listados os perfis de Media existentes e apresentadas as opções de Media.
- No serviço de Imaging é possível consultar as opções de configuração de *imaging* de um *video source*.
- Para o serviço de PTZ é disponibilizado um *streaming* de vídeo através do protocolo RTSP [3]. A URI da *streaming* é obtida através do pedido getStreamURI.

No serviço de Imaging faltam funcionalidades para editar e apresentar as configurações de imagem. Para o serviço de Media falta a funcionalidade para editar as configurações de Media.

O getCapabilities de todos os serviços está a ser feito no serviço de Device Managment e não no respetivo serviço.

A aplicação está desenvolvida para o Android IceCreamsandwish (API *level* 14). A nível de UI os temas desta versão estão bastante desatualizados comparativamente com versões mais recentes. A aplicação apresenta ícones representativos com significados intuitos e contraintuitivos. A nível fragmentos a aplicação é consistente no modo de estruturação. A aplicação usa nos fragmentos de serviço navegação através de *swipes* para apresentar fragmentos adicionais que repartem as funcionalidades. Esta estrutura pode ser melhorada com adição de tabuladores mantendo a navegação por *swipes*.

O sistema de armazenamento de dados da aplicação guarda informações dos pedidos só para uma câmara os quais são apagados quando se troca de câmara. Isto torna-se ineficiente pois há o risco de perder facilmente todos os dados ao executar pequenas tarefas noutras câmaras. A lógica aplicada para verificar os dados existentes era ineficiente e não aplicava boas regras de programação. O armazenamento é feito a nível das atividades e fragmentos e são passados por Intents. As classes dos dados implementavam a interface Serializable para poderem ser passados por Intent. Este aspeto obriga à implementação da interface e não é o mais eficiente. Este fator pode ser resolvido com o armazenamento de dados ao nível da atividade inicial.

A aplicação tem um sistema de armazenamento de informações das câmaras para fornecer acesso às câmaras configuradas sempre que entra na aplicação. O sistema existente guarda os dados em ficheiros de texto sendo que não é ideal para guardar dados deste género. As operações de CRUD sobre um ficheiro de texto obrigam a ler um ficheiro, iterar linha a linha, reescrever por completo o ficheiro em caso de alteração ou adição de informação e, por fim, parar a leitura do mesmo. Um problema é a possibilidade de erro na leitura e escrita do ficheiro que pode corromper dados, sendo preferível uma estrutura de base de dados. Além disso a utilização de um ficheiro de texto para guardar as informações relativas à câmara é ineficiente.

5.2.2 Aplicação para Media

A nível estrutural a aplicação de Media apresenta uma atividade para gestão de câmaras. É possível fazer autenticação através da conta da Google cujos dados são apresentados na barra de navegação lateral. Esta opção obriga à utilização de uma conta de utilizador com a vantagem de apresentar as câmaras configuradas em vários dispositivos. Quando uma câmara é selecionada, é iniciada uma atividade que contém fragmentos para as várias configurações de um perfil de Media: Video Configuration, Audio Configuration, Metadata Configuration, Stream Video, Analytics Configuration e Ptz Configuration. Os valores dos parâmetros apresentados são obtidos recorrendo ao pedido getMediaProfile e getMediaOptions para obter os parâmetros dos perfis de Media e as opções de configuração de um perfil de media respetivamente. Nestes fragmentos é possível reconfigurar as definições de Media. Os valores possíveis para cada parâmetro das várias entidades são obtidos utilizando o pedido getMediaOptions. Para a Stream de vídeo é utilizada a URI RTSP e apresentada numa atividade de Stream.

A nível de funcionalidades e modo de funcionamento é o seguinte:

- Autenticação através da conta Google.

- Gestão de câmaras com recurso a uma base de dados e à barra de navegação lateral.
- Consulta e edição das configurações de vídeo/áudio/ metadados/PTZ.
- Apresentação de um fragmento de Analytics para consulta e edição de dados. A edição e consulta não se encontram implementadas.
- *Streaming* de vídeo.

A aplicação tem uma base de dados SQLLite, nativa do Android para guardar as câmaras configuradas. No modelo de base de dados existem as tabelas "Users" e "Cameras". Os dados de utilizador são os da autenticação da conta Google, nome e email, e os dados das câmaras configuradas para um utilizador são as credenciais, endereço IP, descrição e dados do pedido getDeviceInformation. Existe uma classe, DBcore.java, que implementa as operações de CRUD e as *queries* utilizadas na aplicação.

A aplicação está desenvolvida para o Android Lolipop (API *level* 21). A nível de UI os temas desta estão bastante atualizados sendo que os temas da versão Android Marshmallow (API *level* 23) são semelhantes. Os fragmentos da atividade do serviço Media são consistentes a nível de UI. Na atividade do serviço é possível navegar entre perfis de media através de tabuladores. Nos fragmentos da atividade do serviço de media as informações são divididas em *cards* por categorias.

5.3 Análise

Pretende-se uma aplicação que integre uma arquitetura de comunicação para REST com compatibilidade para outros modos de comunicação, e com funcionalidades de UI para os diversos serviços de um NVT.

A aplicações existentes necessitam de melhorias a nível de UI, funcionalidades e suporte para a versão mais recente do Android (API *level* 23). A versão atual do Android teve melhorias a nível de funcionalidades e segurança, sendo que, há partes da API descontinuadas e desaconselhadas. Pretende-se adicionar à aplicação a biblioteca de comunicação ONIVIF descrita no capítulo anterior.

Para o desenvolvimento da aplicação móvel há possibilidades que implicam diferentes decisões de modo a cumprir os requisitos definidos para a aplicação móvel. É possível a realização de (1) uma nova aplicação, (2) fusão de aplicações ou (3) seguimento de uma das aplicações existentes.

A aplicação com suporte para alguns serviços implementa mais serviços que a aplicação de Media que só se baseia num único serviço. A aplicação com mais serviços necessita de otimizações no código, mudança da arquitetura e atualização da UI para interfaces mais recentes utilizadas no

Android. Por outro lado, a aplicação de Media é mais organizada a nível de interface de utilizador e é implementa as funcionalidades de Media para obter e configurar os perfis. Esta precisa da adição de novos serviços.

A possibilidade mais radical é desenvolver uma aplicação móvel de novo que faz uso de algumas componentes e aplica ideias das aplicações existentes. Esta requer o desenvolvimento de uma nova UI, lógica aplicacional e interligação desta com a arquitetura de comunicação. Há a desvantagem de refazer trabalho já desenvolvido, sendo que, pode trazer melhorias a nível de UI e arquitetura da aplicação.

5.3.1 Justificação da escolha

Pretende-se aproveitar ao máximo o trabalho existente. A aplicação de Media tem uma GUI do serviço Media mais completa e a aplicação com suporte para alguns serviços tem mais fragmentos de serviço. Pretende-se que a nova aplicação implemente todos os serviços. Decidiu-se continuar o desenvolvimento da aplicação com suporte para alguns serviços e posteriormente adicionar os fragmentos do serviço de Media da aplicação de Media. Assim é possível aproveitar as funcionalidades em que as aplicações se destacam. Para efetuar a comunicação com as câmaras é integrada a biblioteca Java REST ONVIF.

5.3.2 Nova Aplicação

Com a fusão das duas aplicações surgiu a necessidade de ajustar a arquitetura aplicacional o que gerou um novo desenho da aplicação.

Nas aplicações analisadas haviam ícones diferentes com o mesmo significado ou com um significado pouco intuitivo. Numa das aplicações a UI estava desatualizada pois apresentava estilos antiquados e não aplicava boas práticas de interface como as definidas pela Google [54]. Assim, redesenhou-se a UI e utilizou-se ícones com significados intuitivos. O poder de processamento e carga nos dispositivos móveis tem vindo a aumentar, no entanto, há dispositivos de menor gama sendo que é necessário gerir os recursos de forma a aumentar a durabilidade da carga e o desempenho da aplicação. No desenho da aplicação é levado em consideração a otimização da mesma para resolver problemas de consumo de carga.

O armazenamento de dados pode ser melhorado, pelo que, pretende-se redesenhar o modo de armazenar os dados fornecidos pela biblioteca Java REST ONVIF. No entanto, a acumulação de dados

das câmaras requere um volume significativo de memória. Devido a este facto é necessário um mecanismo para gerir a quantidade de dados armazenados.

5.4 Desenho da Aplicação

A solução desenhada constitui uma junção das arquiteturas de duas aplicações com melhorias e adições de funcionalidades. O desenho da aplicação, ilustrado na Figura 5.1, tem as camadas GUI, gestão do dados e biblioteca Java REST ONVIF. A GUI é a interface gráfica para o utilizador que fica encarregue de fornecer funcionalidades para gerir as câmaras e serviços. A gestão de dados é a camada que armazena os objetos das classes de dados obtidos pelos métodos da biblioteca Java REST ONVIF e as câmaras configuradas através da GUI.

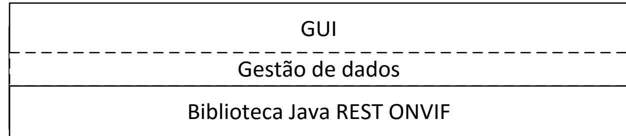


Figura 5.1 – Arquitetura da aplicação

A estrutura da GUI, apresentada na Figura 5.2, define o comportamento da aplicação. Existe uma atividade para o menu inicial de gestão de câmaras que dá acesso à atividade de acesso aos serviços, a qual, implementa uma barra de navegação lateral. Esta última, dá acesso à atividade de *streaming* e aos fragmentos de serviço que em alguns casos têm fragmentos tabuladores. Os fragmentos tabuladores são usados para separar funcionalidades distintas em vários ecrãs de acordo com as categorias de um serviço. A navegação entre serviços tem 8 opções, uma para reproduzir a *stream* de vídeo e as restantes para aceder aos serviços.

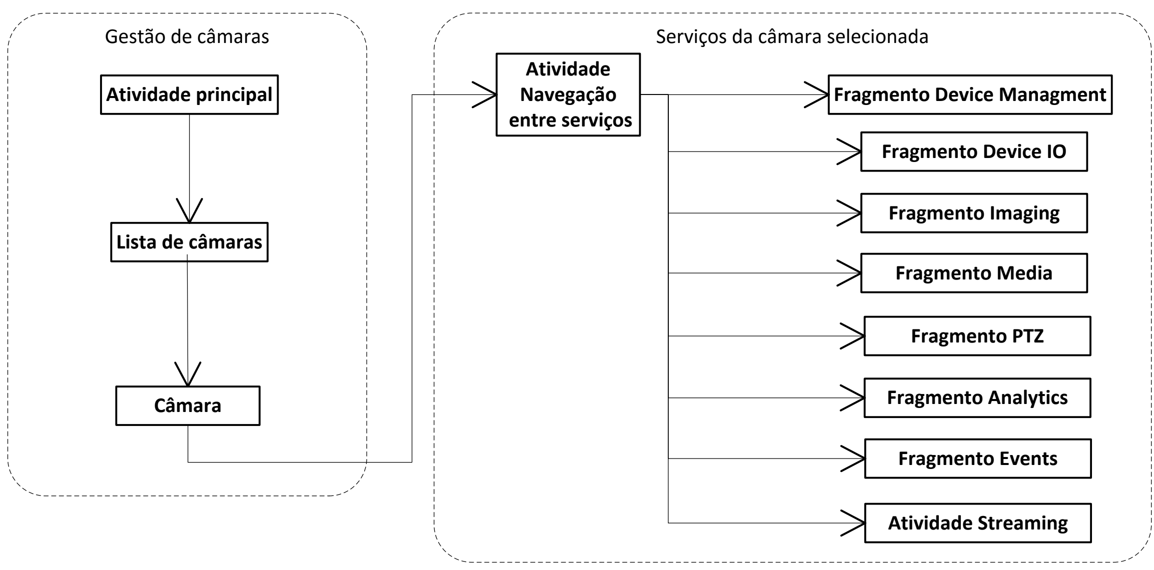


Figura 5.2 – Estrutura da GUI

Os fragmentos dos serviços simples não usam fragmentos de tabuladores. Os fragmentos de serviço simples estão listados na Tabela 5.1.

Tabela 5.1 - Fragmentos de serviço simples

Fragmentos de serviço
Device IO
PTZ
Events
Analytics

Os fragmentos de Serviços e com tabuladores são listados na Tabela 5.2.

Tabela 5.2 - Fragmentos de serviço com fragmentos tabuladores

Fragmentos de Serviço	Fragmentos Tabuladores
Device Managment	System
	Network
	Users
Imaging	Status
	Image Settings
Media	Video
	Audio
	Metadata
	Analytics
	PTZ

À entrada de um fragmento de serviço, quando um dispositivo não suporta alguns dos serviços exibe-se uma notificação através de um Dialog.

Para o serviço Imaging removeu-se os fragmentos de consulta das opções e fez-se a adição de um fragmento com os valores configurados no dispositivo e que permite fazer a alteração dos mesmos respeitando as opções de *imaging* para o respetivo dispositivo. Esta lógica era aplicada na aplicação de Media para obter as Media Options e os Profiles.

Na lógica aplicacional distingue-se o modelo de dados e o modelo da UI de modo a ter independência do tipo da plataforma em uso. Desta forma o modelo de dados mantém-se para outras plataformas devido à independência da plataforma Android. O modelo de dados serve para guardar os objetos de dados dos pedidos realizados à câmara selecionada. Desta forma, os dados podem ser reutilizados quando necessário ou descartados quando desatualizados. Este fator é importante para evitar pedidos redundantes ao servidor diminuindo o volume de tráfego de rede necessário, tanto entre a aplicação e o serviço REST, como entre o serviço e o dispositivo ONVIF. Por exemplo, ao obter um objeto de dados do tipo DeviceInformation este é guardado e associado à câmara utilizada e ao serviço DeviceManagment. Quando se volta a consultar estas informações não é necessário efetuar um novo pedido.

No modelo da UI são aplicadas regras de desenho da interface para tornar a aplicação intuitiva e com lógica de utilização. No desenho da UI tem-se em conta o tipo de dados que se exibe e o tipo de controlos disponibilizados para o utilizador. Por exemplo, para um inteiro com *range* só de *output* é sempre apresentado através de uma SeekBar.

Normalizou-se o modo de apresentar dados (Dados de Saída) e o modo de interagir com os dados (Dados de Entrada) de acordo com o tipo dos mesmos. Desta forma obtém-se uma estrutura de dados coerente e representativa. Os dados de todos as funcionalidades ONVIF suportadas pelo serviço REST são de 5 tipos identificados na primeira coluna da Tabela 5.3. As restantes colunas correspondem aos dados só de entrada, só de saída ou ambos. Para tratar dados de um tipo são utilizadas *views*, sendo algumas personalizadas. Os tipos de *views* a usar dependem do tipo de dados e interação necessária e podem ser consultados na Tabela 5.3, em que as *views* com "*" são personalizados. A SeekBar e RangeSeekBar são personalizadas de forma a mostrar o valor mínimo, máximo e o valor atual. A Checkbox de saída tem texto e ícones representativos de verdadeiro/falso. As *views* de saída personalizadas não podem ser alteradas.

Tabela 5.3 - Views a usar por tipos de interações com os dados

Dados	Só de Entrada	Só de Saída	De Entrada e Saída

Boolean	Checkbox	Checkbox de saída *	Checkboox
Valores numérico ou texto	EditText (por tipo)	TextView	EditText(por tipo)
Enumerados	Spinner	TextView	Spinner
Inteiro pertencente a um intervalo	SeekBar *	SeekBar de saída *	SeekBar *
Par de inteiros pertencentes a um intervalo	RangeSeekBar *	RangeSeekBar de saída *	RangeSeekBar *

O utilizador pode interagir com a aplicação através de gestos e *taps* em ícones. Normalizaram-se gestos e ícones utilizados de acordo com o tipo de ação na interação entre utilizador aplicação. As ações existentes estão listadas na primeira coluna da Tabela 5.4, seguida das colunas de gestos e ícones. Os gestos e ícones são, tipicamente, usados separadamente, não sendo uma obrigação. A normalização do modo de interação com a aplicação é benéfica para o utilizador, uma vez que, torna a aplicação mais intuitiva. Para os ícones podem ser usadas variantes das definidas na terceira coluna, por exemplo, para representar a adição de um utilizador pode-se fundir o símbolo de adicionar com um ícone de um utilizador.

Um exemplo de gesto é ao aplicar um *long tap* numa câmara é apresentado um AlertDialog com opções de editar e eliminar a câmara. Um *tap* numa câmara faz o avanço para o menu de gestão de serviços da câmara.

Tabela 5.4 - Gestos e ícones por ação

Ação	Gesto	Ícone
Consulta	Tap	-
Adicionar	-	+
Editar	Long Tap	
Atualizar	Scroll Down	
Eliminar	Swipe	

5.5 Implementação

Neste capitulo é feita a descrição da implementação dos componentes do desenho. É feita a descrição da estrutura da GUI implementada, interfaces de *input/output*, estado de rede, modos de

armazenar dados da aplicação, mecanismo de gestão de memória, otimizações e normalizações na implementação de fragmentos.

As funcionalidades e resultados finais da aplicação estão documentados no Anexo X – Funcionalidades e Menus da nova aplicação.

5.5.1 Estrutura da GUI

O diagrama de classes com a estrutura das atividades gestão de câmaras e de serviços é ilustrado na Figura 5.3. A classe MainActivity corresponde à atividade principal da aplicação que usa uma ListView que lista um layout personalizado com os dados das câmaras configuradas na aplicação.

Quando um elemento da lista da atividade principal é selecionado inicia-se a atividade da classe NavigationDrawer que contêm fragmentos de serviço, e acesso à atividade de *stream.* A NavigationDrawer dá acesso aos fragmentos através de um painel de navegação lateral com opções de entrada para cada um dos elementos que a constituem. Existem fragmentos simples e fragmentos que contêm fragmentos para implementarem navegação por tabuladores. Existe ainda fragmentos para futuras implementações para os serviços Events e Analytics.

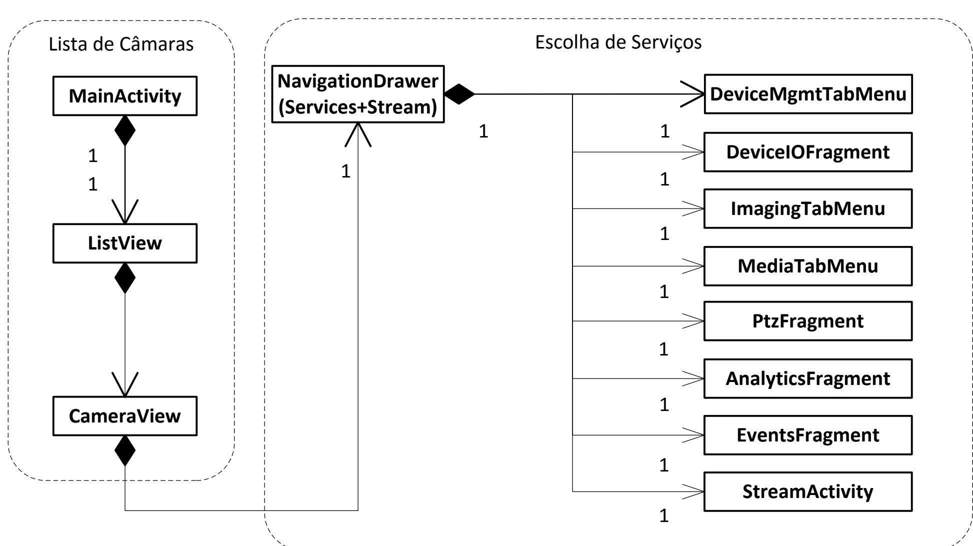


Figura 5.3 - Estrutura das atividades gestão de câmaras e dos serviços

O diagrama de classes para os fragmentos tabuladores de serviço pode ser consultado na Figura 5.4. Para os fragmentos que implementam tabuladores utiliza-se uma classe <A>TabAdapter com um <An>Fragment para cada tabulador do serviço sendo “<A>” o nome do serviço associado e “<An>” o nome representativo de cada fragmento do tabulador.

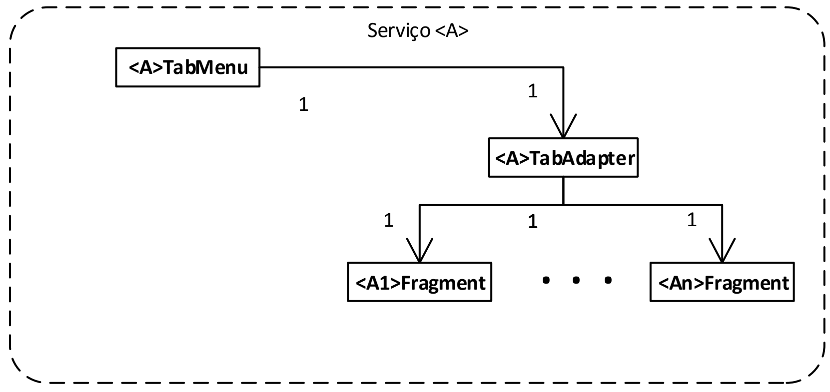


Figura 5.4 - Estrutura para fragmentos de serviço com tabuladores

Os fragmentos de serviço, Adapters e respetivos tabuladores são listado na Tabela 5.5.

Tabela 5.5 - Fragmentos de serviço com tabuladores

Fragmento de serviço	Fragmento Adapter	Fragmentos Tabuladores
DeviceMgmtTabMenu	DeviceMgmtAdapter	SystemFragment SecurityFragment UsersFragment
ImagingTabMenu	ImagingAdapter	StatusFragment SettingsFragment
MediaTabMenu	MediaAdapter	VideoFragment AudioFragment MetadataFragment AnalitycsFragment PtzFragment

Para implementar os tabuladores analisaram-se duas classes FragmentPagerAdapter [55] e FragmentStatePagerAdapter [56] que têm a função de gerir fragmentos tabuladores. Para entender melhor o modo de funcionamento é necessário estudar o ciclo de vida dos fragmentos [57], descrito no Anexo II – Plataforma Android. Os PagerAdapters fazem a gestão da vida dos fragmentos, e o modo de o fazerem é distinto para as duas classes referidas. O uso do FragmentoPagerAdapter implica que seja usado identificadores únicos para os elementos devido a poder haver conflitos, uma vez que, os fragmentos ficam guardados numa *stack* gerida pelo FragmentManager. O FragmentPagerAdapter é melhor quando se tem poucos fragmentos estáticos para serem mostrados. Os fragmentos são guardados em memória pelo que podem ser destruídos quando não são visíveis. Para grandes quantidades de fragmentos este não é recomendado. Para o FragmentStatePagerAdapter o fragmento pode ser completamente destruído mantendo-se o *saved state.* Isto permite usar menos memória associada à vista de cada um dos fragmentos visíveis, mas com o custo de sobrecarga de

processamento quando se troca de fragmentos devido à necessidade de criar fragmentos de novo. O FragmentStatePagerAdapter é útil quando existe uma grande quantidade de fragmentos de tabuladores que não são visíveis. Como a quantidade de fragmentos é baixa em todos os serviços ONVIF, conforme se pode ver n*a* Tabela 5.5, escolheu-se o FragmentPagerAdapter para implementar e gerir os fragmentos de tabuladores.

Para comprovar a escolha e decidir a melhor colocação do código que obtém os dados para colocar na UI, foram feitos testes para verificar quando é que os fragmentos eram criados e quais os métodos invocados para diversas situações do ciclo de vida dos mesmos. Estes testes permitiram também verificar que o FragmentPagerAdapter permite a navegação tanto por *swipe* como por tabuladores. Quando se cria o fragmento que gere os tabuladores verifica-se que são executados os métodos onCreate e onCreateView para o fragmento do tabulador visível e do/s tabuladore/s imediatamente ao lado (por *swipe*). Sempre que um fragmento não esteja criado e seja acedido ou passe a estar ao lado do tabulador visível, é efetuado o onCreate e onCreateView. Depois de ter todos os fragmentos criados só é executado o onCreateView na navegação por fragmentos (por tabuladores ou swipe). Em conclusão, o código que se quer executar quando se retorna a um fragmento do tabulador deve estar presente no onCreateView e as operações da primeira vez que se cria o fragmento devem estar no onCreate.

A nível de alterações na UI foram eliminados textos redundantes e reajustadas as posições dos ícones. Os ícones utilizados foram atualizados de acordo com as normalizações definidas. Os fragmentos que apresentavam informações categorizadas foram divididos em cartões.

Os pedidos do serviço de Media e Imaging requerem um campo para o *profile* e *video source* sendo que é preciso uma opção de seleção dos mesmos. A seleção de um *profile* é feita através de uma partição abaixo da Toolbar que contém o nome do *profile/video source* atual e um ícone a representar a ação de troca de *profiles/video sources*. Na troca de *profile/video source* é utilizado um Dialog com a lista de *profiles/video sources* para seleção.

5.5.2 Interface de input e output

Para exibir avisos como mensagens de erro/sucesso de operações dos métodos são utilizados Toasts. Os pedidos de confirmação, notificações como a falta de conetividade são exibidos através de um Dialog. Os dados *output* relativos as *capabilities* de um serviço e informações *input/output* das definições da aplicação são exibidas através de um Dialog personalizado com *views*. Por exemplo, para editar o limite máximo de armazenamento de dados é utilizado um Dialog com uma SeekBar. Há ainda

Dialogs dinâmicos que podem adicionar e remover *views* consoante as alterações efetuadas. Por exemplo no Dialog Definitions do menu de gestão de câmaras é possível ativar e desativar o armazenamento de dados da aplicação. Quando o armazenamento é ativado é adicionada uma *view* para definir o limite máximo de utilização de memória.

5.5.3 Estado de rede

A aplicação depende essencialmente da comunicação através da rede, pelo que, fez-se mecanismos de armazenamento de dados para reduzir o número de pedidos. Para verificar o estado de rede utilizou-se um BroadcastReceiver por cada atividade da aplicação que tem a função de detetar a ausência de conetividade e notificar o utilizador. A notificação permite aceder as configurações de Wi-Fi de modo a dar a opção de estabelecer uma ligação, ou alterar a atual, para ter acesso à rede.

Quando se configura uma câmara na aplicação é apresentado o estado *online* ou *offline*. Este é obtido quando se cria um objeto da classe OnvifCamera que invoca o getServices e lança a exceção OnvifException em caso de erro. A exceção permite saber a causa do erro e apresentar o estado da câmara.

5.5.4 Sistema de armazenamento de dados

Os objetos de dados da biblioteca são guardados pela aplicação com objetivo de minimizar os pedidos de rede efetuados entre os vários componentes do sistema: aplicação, servidor e câmara. Este fator é importante para melhorar o desempenho da aplicação e diminuir o consumo de dados. De modo a gerir a memória consumida implementou-se um sistema de gestão da mesma responsável por libertar espaço quando a memória consumida ultrapassa um limite definido.

Assim, são guardados os objetos das classes de dados fornecidas pela biblioteca Java REST ONVIF por câmara. Os objetos de dados são guardados em listas por câmara, por serviço, tipos de dados, *media profiles* e *video sources* de acordo com os métodos usados. Para isso são usados enumerados para os parâmetros conhecidos, nomeadamente, os serviços e os tipos de dados. Quando é necessário fazer corresponder dados a um *video source* ou *media profile* usa-se o *token* dos mesmos. Os dados são guardados numa lista em que o índice corresponde a uma câmara configurada na aplicação. Esta estrutura mantém a lógica de utilização de listas de dados da aplicação dos serviços, mas acrescenta suporte para guardar mais dados e os enumerados que permitem melhorar a lógica da aplicação. Esta estrutura é desenhada de modo a ser independente da UI. Desta forma é

possível usar a mesma estrutura de dados para outras aplicações e plataformas sem ser necessário redesenhar a estrutura de dados. No entanto a UI é dependente da estrutura de dados.

5.5.5 Base de dados de câmaras

Para reconfigurar câmaras sempre que se entra na aplicação usou-se uma base de dados. O Android oferece uma base de dados SQL local [58]. A base de dados SQL criada tem uma tabela com os dados correspondentes aos campos da classe OnvifCamera da Biblioteca Java REST ONVIF ou seja: o nome da câmara, endereço IP, nome de utilizador, palavra-chave e endereço do servidor REST. O modelo de base de dados utilizado pode ser consultado na Figura 5.5. A base de dados só pode ser usada pela aplicação evitando que outras aplicações acedam à mesma [58].

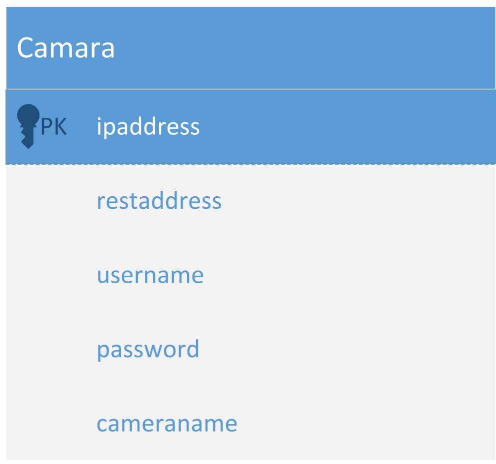


Figura 5.5 - Modelo de base de dados para câmaras

5.5.6 Gestão de memória

Em tempo de execução a aplicação guarda os objetos das classes de dados que vão sendo obtidos das câmaras selecionadas. Como este fator contribui para o aumento do uso de memória analisou-se o consumo da mesma para criar um sistema de gestão de memória. Pretende-se assim libertar memória em uso quando esta ultrapassar um determinado valor. Como os dados variam de câmara para câmara, por exemplo, número de perfis, o valor limite em bytes foi obtido com base em alguns testes. Para isso, fizeram-se testes para obter a memória utilizada para guardar todos os dados necessários à aplicação. A medição foi realizada utilizando o Android Studio, que fornece ferramentas de análise de consumo de memória da aplicação, sendo possível obter a memória em uso e libertada.

O primeiro teste consiste em ver qual a memória consumida pela aplicação sem fazer qualquer pedido, em que se obteve aproximadamente 15 MB de memória reservada, cujo gráfico pode ser

consultado na Figura 5.6. Verifica-se que a aplicação libertou 8.30MB de recursos após 5 segundos do início da mesma.

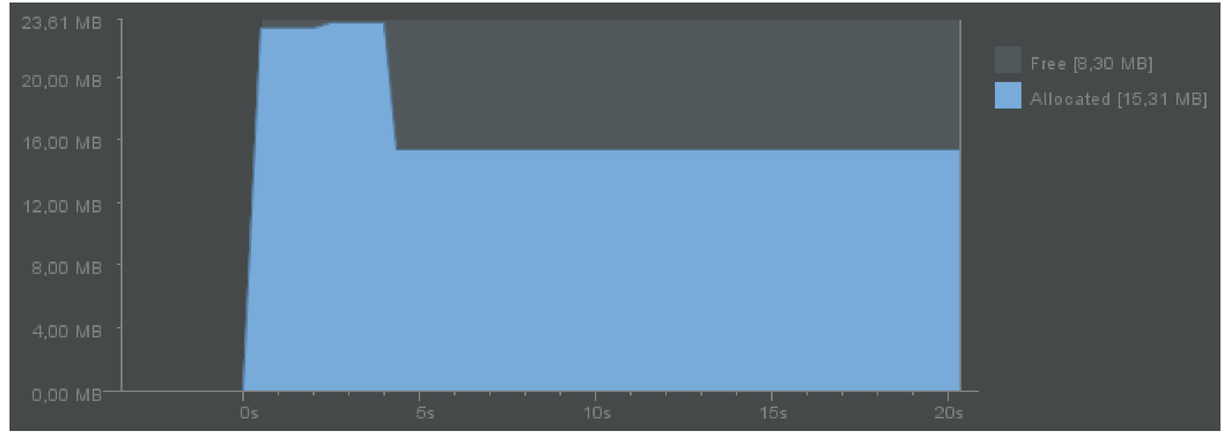


Figura 5.6 - Memória libertada e alocada pela aplicação sem efetuar pedidos

No segundo teste efetuou-se pedidos que são guardados em memória e obteve-se aproximadamente 25 MB de memória alocada cujo gráfico é exibido na Figura 5.7. A quantidade de dados dos diferentes pedidos pode variar significativamente, mas isso não é importante, pois estes valores são utilizados apenas como referência para estimar valores limite da memória utilizada para guardar dados das câmaras.

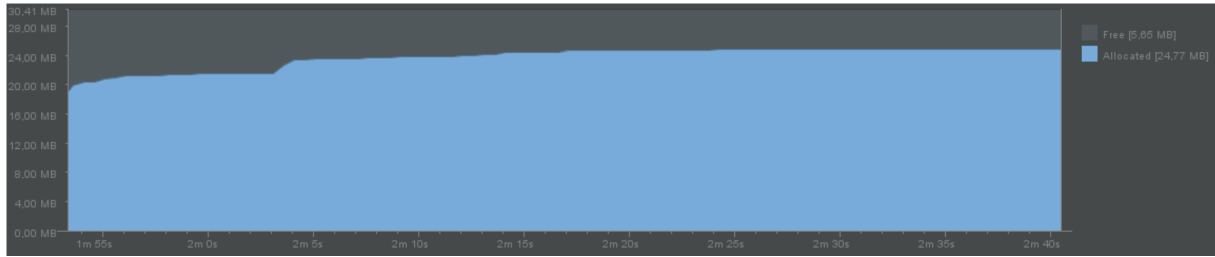


Figura 5.7 - Memória libertada e alocada pela aplicação depois de efetuar diversos pedidos

Verifica-se que a aplicação utiliza cerca de 10MB (25MB – 15MB) de memória para armazenar todos os dados da câmara HIKVISION, modelo “DS-2CD2612F-I” e *firmware* “V5.3.0 build 151016” utilizada nos testes.

Optou-se por definir um valor para o máximo de 128 MB (cerca de dez vezes superior à alocada pelos pedidos e próximo de potência de base 2), e também de dar ao utilizador a opção de desligar o armamento de dados. Esta funcionalidade é importante para dispositivos que tenham menos capacidade dando ao utilizador a opção de minimizar o consumo de memória. Por outro lado, quanto menor for a memória alocada permitida mais pedidos serão realizados, dependendo de como se utiliza a aplicação.

5.5.7 Toolbar

A barra de ferramentas usada na aplicação com alguns serviços é a ActionBar da biblioteca AppCompat o que implica a dependência da mesma e um menor controlo nas configurações de UI por defeito, que podem mudar dependendo da versão do Android. Na aplicação de Media é usada a Toolbar que têm um *design* mais atual. A Toolbar é uma alternativa que tal como a ActionBar fornece aos desenvolvedores modos de adicionar opções de menu, tabuladores, entre outras funcionalidades. Uma vantagem da Toolbar é que é muito personalizável podendo mesmo ser colocada em qualquer lugar da interface de utilizador. Algumas das tarefas que se podem fazer é adicionar botões de navegação, logótipos, título, subtítulo, itens de menu e *views* personalizadas [59]. A personalização da barra de ferramentas é essencial para evitar problemas com as mudanças dos temas dependendo da versão do Android, fornecendo maior controlo na interface de utilizador.

Não se alterou o local da barra de ferramentas, mas foram adicionadas opções de menu. Há opções comuns na Toolbar para os serviços como o *update* que serve para atualizar as informações manualmente e a consulta das *capabilities*. Em situações reais esta opção é muitas vezes usada pelas aplicações existentes porque é eficaz e permite ao utilizador decidir quando se quer atualizar os dados.

No menu do serviço de Imaging, a Toolbar da Figura 5.8, são disponibilizadas opções para guardar, mostrar um *snapshot* e opções ocultas nos três pontos para atualizar dados, guardar alterações e mostrar as *capabilities*. Para os serviços a Toolbar tem um símbolo com três barras sobrepostas para aceder à barra de Navegação lateral.

Figura 5.8 - Toolbar do serviço Imaging

O menu inicial para gestão de câmaras apresenta a Toolbar da Figura 5.9 com opções para atualizar dados, consultar o About, ver e editar as definições, e eliminar dados.

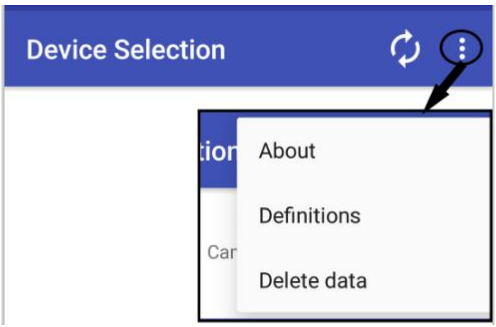


Figura 5.9 - Toolbar da atividade de gestão de câmaras

O serviço de Media também apresenta uma opção para guardar as alterações efetuadas.

5.5.8 Otimização no uso de recursos

De modo a otimizar a utilização de recursos usados pela aplicação aplicaram-se regras de *design* e utilizaram-se ferramentas para minimizar o consumo de memória de *bitmaps*. Os recursos usados são Strings, cores, dimensões e *drawables* e *layouts* e são exteriorizados do código. A utilização destes recursos tem vantagens como facilidade na edição de campos de texto, suporte de várias línguas, troca de cores, dimensões e de ícones. De modo a suportar diferentes configurações tendo em conta as características do dispositivo os recursos encontram-se na diretoria "res/" do projeto da aplicação que tem subdiretorias de acordo com o tipo e configuração. Existem recursos por defeitos que são usados independentemente do dispositivo e recursos alternativos para serem utilizados por dispositivos com características específicas.

Para títulos fixos definiu-se *strings* no ficheiro "Strings.xml". Atribuiu-se o código RGB "#424242", uma cor cinzenta, para os ícones do menu e para alguns campos de texto. Gerou-se imagens com diversas resoluções de ecrãs de modo a preservar a qualidade de imagem para diferentes resoluções sem alocar memória desnecessária. Os ícones foram essencialmente obtidos no repositório de ícones da Google [60] e editados através da ferramenta paint.NET [61]. Para geração de ícones para diferentes resoluções de ecrã foi utilizado o Android Asset Studio [62] que permite a correção de cores pelo código RGB e definir parâmetros como o *trim/padding/size* do ícone. Fragmentou-se as semelhanças dos *layouts* criados em diferentes ficheiros de modo a evitar repetição.

5.5.9 Normalizações para os menus de serviço

Os fragmentos dos serviços permitem consultar informações e alterar configurações das câmaras. Estes estão responsáveis pela realização da maior parte dos pedidos ao servidor. Os dados dos pedidos são armazenados na aplicação conforme descrito em 5.5.4.

Existem pedidos que devem ser repetidos caso a informação possa estar desatualizada. Por exemplo o pedido getSystemDateAndTime do serviço Device Management tem algumas informações que são facilmente desatualizadas pelo que deve ser repetido sempre que se consulta a Data e Hora. Os restantes pedidos podem ser guardados normalmente na aplicação.

Quanto ao envio de alterações para as câmaras existem pedidos que podem ter dados num único fragmento ou distribuídos por vários fragmentos, e existem fragmentos que podem conter várias operações. Desta forma, normalizou-se o modo de interação com o utilizador de acordo com a relação entre operações e fragmentos. As possibilidades existentes e tipo de comportamento da aplicação são divididos nas quatro situações seguintes:

1. **Operações diretas (só podem existir num único fragmento):** (e.g., eliminar um utilizador) em certos casos as operações que se querem fazer são diretas pois não implicam estar a configurar vários dados. Sendo assim o utilizador quando realiza uma operação deste género sabe que só esta a fazer uma alteração. As operações diretas só têm uma ação na UI e só estão presentes num fragmento. Estas podem ser realizadas diretamente sem ineficiências nem a necessidade de um botão de guardar dados. Nestes casos é emitida uma mensagem para confirmação da operação a realizar. Se o pedido for efetuado e tiver sucesso as novas informações são armazenadas na aplicação caso necessário.
2. **Operações com dados distribuídos por vários fragmentos:** (e.g., SetProfile do serviço Media). Quando se faz alguma alteração dos parâmetros num dos fragmentos, o botão de guardar da Toolbar torna-se visível. Se à saída do menu de serviço os dados não tiverem sido guardados é exibida uma notificação para decidir se quer guardar as alterações ou ignorar. Caso o utilizador guarde as novas informações é feito o pedido ao servidor de alteração de dados e caso seja efetuado com sucesso os novos dados são guardados na aplicação se necessário.
3. **Relação de 1:1 entre operação divisível e fragmento:** (e.g., operação setSystemDateAndTime do serviço Device Managment). Este caso é aplicado para operações que sejam divisíveis, ou seja, tem várias ações na UI para a realização de um pedido num só fragmento. Tal como na situação anterior, o botão de guardar é exibido, assim que exista alguma alteração. No entanto o pedido de confirmação para guardar informações é feito à saída do fragmento caso exista alguma alteração pendente (é feita a verificação quando se faz *swipe* ou se acede a um tabulador diretamente).

Para os casos acima enunciados, a lógica aplicada é amigável para o utilizador e aplicação, evitando fazer pedidos desnecessários sempre que se efetua alguma alteração. Assim, é evitado sobrecarregar a rede com pedidos redundantes. A interface de utilizador muda as informações apresentadas consoante o sucesso dos pedidos assim como os dados armazenados. Alguns comportamentos são muito comuns em aplicações atuais e são usados porque melhoram o funcionamento das aplicações e são amigáveis do utilizador.

O *listener* de erro é usado para apresentar informações de erro e de insucesso da operação, o *listener* de progresso é usado para verificar o progresso e sucesso da operação caso necessário e o *listener* de resultado é usado para recolher e guardar os objetos de dados obtidos nos pedidos.

As alterações de dados num fragmento são detetas através de um *listener* dos dados de input que pode ter nomes diferentes dependendo do tipo de dados. Este *listener* implementa a ação onChange, por exemplo, para um EditText tem-se o método "onTextChanged". Quando existe uma alteração coloca-se o estado de hasChanges a verdadeiro. Este valor passa a falso quando são guardadas informações ou quando o utilizador não quer guardar as alterações.

Para qualquer caso, se forem guardadas as alterações é efetuado um pedido ao servidor e em caso de sucesso os dados são armazenados na aplicação caso necessário.

Os métodos utilizados na aplicação correspondentes aos três casos acima normalizados estão listados na Tabela 5.6.

Tabela 5.6 - Métodos usados de acordo com as normalizações.

Métodos do 1º caso	Métodos 2ºcaso	Métodos 3º caso
setSystemDateAndTime	setProfiles	setSystemDateAndTime
deleteScope	setImageSettings	
addScope		
editScope		
addUser		
editUser		
deleteUser		
setNTP		
setDNS		
setDynDNS		

5.5.10 Fragmentos do serviço Media e Imaging

Os fragmentos do serviço Media e Imaging têm particularidades em relação aos restantes fragmentos de serviço.

No Media, antes entrar no fragmento do serviço é efetuado o pedido para obter os *profiles* para seleção. São ainda carregadas as *options* e *settings* de *media* para apresentar os dados da câmara e as restrições de configuração dos mesmos. A aplicação de Media existente estava bastante completa, pelo que, trocou-se o tipo de comunicação sem a necessidade de fazer pedidos extra. O fragmento de Imaging tem a mesma abordagem na seleção de *video sources.*

Para o Media, reparou-se que os dados da UI que eram opcionais e não implementados pelo dispositivo estavam a ser apresentados sem qualquer tipo de valor existente. Desta forma, aplicou-se a lógica de esconder as Views para os dados opcionais que não fossem suportados pelo dispositivo,

garantindo que o utilizador não seja induzido em erro ao configurar as definições de *media.* O fragmento de Imaging segue esta lógica.

6. CONCLUSÃO

6.1 Resultados

Um dos objetivos delineados era o estudo e melhoramento do serviço REST ONVIF. Conseguiu-se introduzir melhorias na configuração do serviço assim como se atualizou a configuração do servidor HTTP para a versão mais recente. Os pedidos getServices e getCapabilities de cada serviço ONVIF foram adicionados ao serviço REST. Os testes realizados permitiram verificar que se conseguiu obter os serviços suportados pelo dispositivo ONVIF através do pedido getServices e para o getCapabilities conseguiu-se obter as *capabilities* do serviço respetivo. Identificaram-se erros que foram corrigidos, tais como, tratamento de valores booleanos como *strings* ou inteiros. Foram realizados testes individuais ao servidor utilizando a ferramenta Postman para verificar as correções e adições implementadas.

Pretendia-se desenhar e desenvolver uma biblioteca para comunicação através de REST com câmaras ONVIF. Desenvolveu-se a biblioteca Java para abstrair o serviço REST com capacidade para suportar outros modos de comunicação e que: tem uma estrutura alinhada com o ONVIF, permite configurar câmaras, suporta pedidos do serviço REST ONVIF, permite adicionar facilmente outros modos de comunicação, tem comunicação segura, permite comunicação de modo síncrono e assíncrono, disponibiliza os dados ONVIF em Java e fornece informações de progresso, resultados e erros. A biblioteca suporta a configuração de NVTs assim como os seus serviços ONVIF. Testaram-se experimentalmente os diversos componentes da biblioteca e conseguiu-se realizar com sucesso todas as operações ONVIF suportadas pelo serviço REST (documentadas no Anexo VI – Classes de Serviço). A documentação da biblioteca é extensa pelo foi feita em Javadoc, de modo a fornecer ao programador uma API documentada.

Delineou-se a restruturação de uma nova aplicação Android com integração de funcionalidades suportadas por duas aplicações existentes. A aplicação tem como requisito principal implementar as funcionalidades da biblioteca Java desenvolvida. Pensou-se numa nova estrutura para a aplicação que aproveitou trabalho já desenvolvido neste âmbito. A aplicação tem um menu para gestão de câmaras, menus para todos os serviços de um NVT, menu para visualização de uma *stream* de vídeo, atualizaram-se funcionalidades na Toolbar e fez-se uma nova barra de navegação lateral para navegação entre serviços. Além da integração, foram introduzidas e testadas novas funcionalidades como por exemplo as configurações de imagem. Melhorou-se a UI através da normalização de

fragmentos, *views,* gestos e ícones. A aplicação recorre ao *material design* e compre regras de desenho da Google. Os serviços mais complexos têm menus com tabuladores para navegar entre vários fragmentos do mesmo. A utilização de recursos foi melhorada através da adoção de um gestor de fragmentos mais eficiente. A aplicação disponibiliza todas as funcionalidades da biblioteca Java REST ONVIF exceto as do serviço PTZ, porque o dispositivo de teste não suporta. Para além dos testes individual do servidor e da biblioteca com o servidor, foram realizados testes experimentais de integração da aplicação, tendo sido obtido sucesso em todas as funcionalidades implementadas (ver Anexo X – Funcionalidades e Menus da nova aplicação).

6.2 Conclusões

Conforme descrito na seção anterior, foi desenvolvido um trabalho variado, quer em termos conceptuais quer em termos tecnológicos, e volumoso. Todos os objetivos foram atingidos e este trabalho contribuiu com avanços muito significativos relativamente aos anteriores.

No serviço REST ONVIF foram adicionadas funcionalidades que seguem a segunda versão das normas ONVIF, nomeadamente para obter os serviços e as *capabilities* separadamente, e que têm impacto na implementação da biblioteca. Verificou-se que a parte das respostas com recursos e verbos para utilização da API do serviço tinha conteúdos incompletos, por exemplo ao pedido do getProfiles, o que dificultou o desenvolvimento da biblioteca. Para além disso, as respostas não separam informação da API dos dados ONVIF. Este aspeto minimiza o desempenho uma vez que as mensagens têm informação extra que pode não ser usada pelas aplicações cliente.

Uma das contribuições fundamentais deste trabalho é a biblioteca Java REST ONVIF. Esta tem uma dimensão grande, com a serialização e desserialização de quase duas centenas de classes de dados, classes de serviço com operações síncronas e assíncronas num total de métodos superior a uma centena, e diversos *handlers.* A biblioteca tem um desenho OO complexo, composto por uma classe câmara, interfaces serviço, classes abstratas e concretas por serviço (para suportar diversos modos de comunicação) e um sofisticado mecanismo de informação qualitativa sobre o progresso dos pedidos. A biblioteca fornece uma API simples, bastando instanciar um objeto câmara que disponibiliza os serviços suportados e respetivas funcionalidades de forma estruturada e direta.

Outra das principais contribuições deste trabalho é a aplicação Android para dispositivos NVT. Esta aplicação suporta a maior parte das funcionalidades desse tipo de dispositivos e é perfeitamente funcional. Por isso, pode ser utilizada para configurar qualquer câmara que suporte a norma ONVIF. A

aplicação foi dotada de características não funcionais importantes como o armazenamento de dados com mecanismo gestão de memória. Esta característica permitiu reduzir o número de pedidos e a utilização de um gestor que não elimina dinamicamente fragmentos permitiu também obter reduções de uma forma simples. No entanto, devido à existência de vários controlos UI para os dados de um mesmo pedido e de vários pedidos num único fragmento, a minimização do número de pedidos ao serviço REST torna-se muito complexa.

6.3 Trabalho Futuro

A evolução deste trabalho abre novas frentes para continuação do tema. A forma alternativa de comunicação foi prevista no desenvolvimento da biblioteca Java ONVIF e poderá agora ser concretizada. Nomeadamente a implementação através de uma biblioteca JNI que faz a comunicação direta com câmaras ONVIF já foi iniciada e poderá ser facilmente integrada na biblioteca. Outras funcionalidades poderão ser adicionadas, como os serviços Events e Analytics, esta última quando estiver disponível no serviço REST ou noutra interface.

Relativamente ao serviço REST, há respostas ONVIF completadas e há funcionalidades por implementar que podem ser adicionadas (como, por exemplo, um Discovery Proxy para descoberta de câmaras em redes remotas). Existe também um protótipo do serviço Events que propaga os eventos utilizando o protocolo Server Sent Events, e que poderá ser integrado. Mas, o destaque vai para as possibilidades de melhorar a API uma vez que um dos objetivos do serviço é reduzir a quantidade de dados das mensagens e a informação da API é em alguns casos volumosa e maior que o conteúdo ONVIF das respostas, propõem-se a separação dessa informação em novos pedidos e respostas, utilizando outros métodos HTTP. Propõe-se também a adoção de um novo modelo de dados para essa informação.

Quanto à aplicação podem ser adicionadas funcionalidades em conformidade com a evolução da biblioteca Java REST ONVIF. Dos serviços implementados na aplicação, o PTZ, é aquele com menos funcionalidades na UI. Deste modo, pode-se integrar as funcionalidades de *pan, tilt* e *zoom* na *stream* de vídeo, que deve ser testado num dispositivo que suporte PTZ. A aplicação tem locais para adicionar os serviços de Analytics e Events quando estiverem disponíveis na biblioteca. Estas adições permitem aumentar a riqueza da aplicação tornando-a completa em termos de serviços ONVIF NVT.

As aplicações de videovigilância são usadas muitas vezes em vários dispositivos e algumas utilizações englobam a visualização de vídeo de várias câmaras. O futuro da aplicação Android pode

assim passar pelo suporte multiplataforma e a reprodução de vídeo de diversas câmaras em simultâneo.

Os sistemas de videovigilância são bastantes úteis em aplicações Android para permitir trabalhar em qualquer lugar, mas também existem aplicações para computadores pessoais ou fixos. Além de uma aplicação *web* já existente, pode ser criada uma aplicação *desktop* para qualquer sistema operativo com base na biblioteca Java ONVIF. Para isso, apenas será necessário adaptar algum código que é específico de Android (como a AsyncTask) para tornar a biblioteca compatível com J2SE.

BIBLIOGRAFIA

[1] "ONVIF Specification Map." [Online]. Available: http://www.onvif.org/specs/DocMap.html. [Accessed: 27-Jan-2016].

[2] "ONVIF Network Video Transmitter Device Definition."

[3] R. F. T. Nogueira, "'Aplicação Android para configuração e acesso direto a câmaras ONVIF'," Universidade do Minho, 2014.

[4] H. F. da S. Dias, "Hélder Filipe da Silva Machado Dias Comunicação Android com câmaras ONVIF e gestão de Perfis de Media," 2015.

[5] S. F. Lopes, S. Silva, J. Mendes, J. C. Metrolho, and D. Duque, "Development of a library for clients of ONVIF video cameras : challenges and solutions," *Ind. Technol. (ICIT), 2013 IEEE Int. Conf.*, pp. 1260–1266, 2013.

[6] "ONVIF IP Camera Monitor." [Online]. Available: https://play.google.com/store/apps/details?id=net.biyee.onvifer. [Accessed: 09-Jan-2016].

[7] "IP Cam Viewer Basic." [Online]. Available: https://play.google.com/store/apps/details?id=com.rcreations.ipcamviewerBasic. [Accessed: 12-Jan-2016].

[8] "tinyCam Monitor FREE." [Online]. Available: https://play.google.com/store/apps/details?id=com.alexvas.dvr. [Accessed: 09-Jan-2016].

[9] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study," *Scenario*, vol. 59715, pp. 1–3, 2009.

[10] O. N. V. T. Definition, "ONVIF ™ Network Video Transmitter Device Definition," pp. 1–7, 2011.

[11] "IP Camera Standards Use Revealed." [Online]. Available: http://ipvm.com/reports/ip-camera-standards-deployment. [Accessed: 25-Feb-2016].

[12] L. Richardson and S. Ruby, *RESTful web services*. 2007.

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

[14] H. Zhao and P. Doshi, "Towards Automated RESTful Web Service Composition Haibo Zhao and Prashant Doshi," 2009.

[15] "ONVIF Device Manager Network video device management software." [Online]. Available: https://sourceforge.net/p/onvifdm/wiki/Home/. [Accessed: 04-Oct-1BC].

[16] "Happytimesoft - Onvif Client Library." [Online]. Available: http://www.happytimesoft.com/products/onvif-client-for-android/index.html. [Accessed: 03-Oct-1BC].

[17] "onvif-java-lib." [Online]. Available: https://github.com/milg0/onvif-java-lib. [Accessed: 04-Oct-1BC].

[18] "python-onvif." [Online]. Available: https://github.com/quatanium/python-onvif. [Accessed: 03-Oct-1BC].

[19] "onvifcpplib." [Online]. Available: https://github.com/veyesys/onvifcpplib. [Accessed: 04-Oct-1BC].

[20] "The gSOAP Toolkit for SOAP and REST Web Services and XML-Based Applications." [Online]. Available: http://www.cs.fsu.edu/~engelen/soap.html. [Accessed: 04-Oct-1BC].

[21] "Client to ONVIF NVT devices Profile S: cameras." [Online]. Available: https://www.npmjs.com/package/onvif. [Accessed: 04-Oct-1BC].

[22] "ONVIF 2.0 Service Operation Index." [Online]. Available: http://www.onvif.org/onvif/ver20/util/operationIndex.html. [Accessed: 05-Oct-1BC].

[23] "tPacketCapture." [Online]. Available: https://play.google.com/store/apps/details?id=jp.co.taosoftware.android.packetcapture&hl=pt_PT. [Accessed: 10-Jan-2016].

[24] "ksoap2-android Project." [Online]. Available: http://simpligility.github.io/ksoap2-android/index.html. [Accessed: 25-Oct-1BC].

[25] J. F. Lopes, "Aplicação web para configuração e acesso a câmaras ONVIF," 2013.

[26] A. Alexandra and O. Passos, "Reconfiguração e novas funcionalidades para um servidor REST ONVIF : serviço de IO e configuração de descoberta," 2015.

[27] "Apache HTTP server project." [Online]. Available: https://httpd.apache.org/. [Accessed: 12-Sep-2016].

[28] "Postman." [Online]. Available: https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop. [Accessed: 10-Sep-1BC].

[29] "Putty Download Page." [Online]. Available: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. [Accessed: 28-Feb-2016].

[30] C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," pp. 1–30, 2006.

[31] “WinSCP free SFTP, SCP and FTP client for Windows.” [Online]. Available: https://winscp.net/eng/index.php. [Accessed: 20-Sep-2016].

[32] “Upgrading to 2.4 from 2.2.” [Online]. Available: https://httpd.apache.org/docs/2.4/upgrading.html. [Accessed: 28-Jan-2016].

[33] “AsyncTask.” [Online]. Available: http://developer.android.com/reference/android/os/AsyncTask.html. [Accessed: 10-Nov-2016].

[34] “Processes and Threads.” [Online]. Available: http://developer.android.com/guide/components/processes-and-threads.html. [Accessed: 10-Oct-2016].

[35] “Communicating with the UI Thread.” [Online]. Available: http://developer.android.com/training/multiple-threads/communicate-ui.html. [Accessed: 10-Oct-2016].

[36] “Unchecked Exceptions — The Controversy.” [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html. [Accessed: 08-Feb-2016].

[37] E. Gamma, R. Heim, R. Johnson, and J. Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.

[38] J. Paul, “Why character array is better than String for Storing password in Java.” [Online]. Available: http://javarevisited.blogspot.pt/2012/03/why-character-array-is-better-than.html. [Accessed: 04-Dec-2015].

[39] “Performance Tips.” [Online]. Available: http://developer.android.com/training/articles/perf-tips.html. [Accessed: 03-Dec-2015].

[40] “Performance improvement techniques in Object creation.” [Online]. Available: http://www.precisejava.com/javaperf/j2se/Objects.htm. [Accessed: 03-Dec-2015].

[41] J. Bloch, “Avoid creating unnecessary objects.” [Online]. Available: http://www.informit.com/articles/article.aspx?p=1216151&seqNum=5.

[42] “Class ProtocolException.” [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/net/ProtocolException.html. [Accessed: 10-Mar-1BC].

[43] “Class IOException.” [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html. [Accessed: 10-Mar-

1BC].

[44] "JSONException." [Online]. Available: https://developer.android.com/reference/org/json/JSONException.html?hl=ko. [Accessed: 10-Mar-1BC].

[45] "Android Studio The Official IDE for Android." [Online]. Available: https://developer.android.com/studio/index.html. [Accessed: 13-Sep-2016].

[46] "Speed up your development lifecycle with an easy, accessible, and effective testing and collaboration tool." [Online]. Available: https://www.genymotion.com/.

[47] "Introducing Xamarin Android Player." [Online]. Available: https://www.xamarin.com/android-player. [Accessed: 13-Mar-2016].

[48] "Connecting to the Network." [Online]. Available: http://developer.android.com/training/basics/network-ops/connecting.html. [Accessed: 10-Oct-2016].

[49] "Android 6.0 Changes." [Online]. Available: http://developer.android.com/about/versions/marshmallow/android-6.0-changes.html#behavior-apache-http-client. [Accessed: 10-Jan-2016].

[50] B. Venners, "Exceptions in Java The full story of exceptions in the Java language and virtual machine," 1998. [Online]. Available: http://www.javaworld.com/article/2076700/core-java/exceptions-in-java.html?page=3.

[51] B. Woolf, R. Martin, D. Riehle, and F. Buschmam, *Pattern Languages of Program Design*, vol. 3. 1998.

[52] "Class System." [Online]. Available: http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html. [Accessed: 04-Apr-2016].

[53] "How to Write Doc Comments for the Javadoc Tool." [Online]. Available: http://www.oracle.com/technetwork/articles/java/index-137868.html. [Accessed: 14-Sep-2016].

[54] "Up and running with material design." [Online]. Available: https://developer.android.com/design/index.html. [Accessed: 17-May-1BC].

[55] "FragmentPagerAdapter." [Online]. Available: https://developer.android.com/reference/android/support/v4/app/FragmentPagerAdapter.html. [Accessed: 28-Dec-1BC].

[56] "FragmentStatePagerAdapter." [Online]. Available: https://developer.android.com/reference/android/support/v4/app/FragmentStatePagerAdapter.html. [Accessed: 28-Dec-1BC].

[57] "Fragments." [Online]. Available: https://developer.android.com/guide/components/fragments.html. [Accessed: 23-Nov-1BC].

[58] "Saving Data in SQL Databases." [Online]. Available: https://developer.android.com/training/basics/data-storage/databases.html. [Accessed: 02-Jul-1BC].

[59] J. Montemagno, "Android Tips: Hello Toolbar, Goodbye Action Bar," 2014. [Online]. Available: https://blog.xamarin.com/android-tips-hello-toolbar-goodbye-action-bar/ . [Accessed: 08-May-1BC].

[60] "Material icons." [Online]. Available: https://design.google.com/icons/. [Accessed: 01-Jun-1BC].

[61] "Paint.NET." [Online]. Available: http://www.getpaint.net/index.html. [Accessed: 01-Jun-1BC].

[62] "Android Asset Studio." [Online]. Available: https://romannurik.github.io/AndroidAssetStudio. [Accessed: 01-Jun-1BC].

[63] "Application Fundamentals." [Online]. Available: https://developer.android.com/guide/components/fundamentals.html#Components. [Accessed: 24-Sep-1BC].

[64] "Activities." [Online]. Available: https://developer.android.com/guide/components/activities.html. [Accessed: 24-Sep-1BC].

[65] "Services." [Online]. Available: https://developer.android.com/guide/components/services.html . [Accessed: 24-Sep-1BC].

[66] "Content Providers." [Online]. Available: https://developer.android.com/reference/android/content/ContentProvider.html. [Accessed: 24-Sep-1BC].

[67] "BroadcastReceiver." [Online]. Available: https://developer.android.com/reference/android/content/BroadcastReceiver.html. [Accessed: 24-Sep-1BC].

[68] "View." [Online]. Available: https://developer.android.com/reference/android/view/View.html. [Accessed: 24-Sep-1BC].

[69] "Layouts." [Online]. Available: https://developer.android.com/guide/topics/ui/declaring-

layout.html. [Accessed: 24-Sep-2016].

[70] "Intent." [Online]. Available: https://developer.android.com/reference/android/content/Intent.html. [Accessed: 24-Sep-1BC].

[71] "Resources Overview." [Online]. Available: https://developer.android.com/guide/topics/resources/overview.html. [Accessed: 24-Sep-1BC].

[72] "App Manifest." [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro.html. [Accessed: 24-Sep-1BC].

[73] "Introducing JSON." [Online]. Available: http://www.json.org/. [Accessed: 07-Oct-1BC].

[74] "Ubuntu Apt-Get Documentation." [Online]. Available: https://help.ubuntu.com/lts/serverguide/apt-get.html. [Accessed: 03-Dec-2015].

[75] "Saving Key-Value Sets." [Online]. Available: https://developer.android.com/training/basics/data-storage/shared-preferences.html. [Accessed: 04-May-1BC].

[76] "SharedPreferences." [Online]. Available: https://developer.android.com/training/basics/data-storage/shared-preferences.html. [Accessed: 04-May-2016].

ANEXO I – CONFIGURAÇÕES DO SERVIDOR WEB REST

Ficheiro ssl-rest-onvif.pt.conf da diretoria do apache "/sites-available" ou "sites-enable".

```
<VirtualHost _default_:443>
			ServerAdmin RicardoCOPeixoto@gmail.com
			ServerName restonvifricardo
			AllowEncodedSlashes On

			SSLProtocol all
			SSLEngine on
			SSLCertificateFile /etc/ssl/private/restonvifricardo.crt

# Necessario para as paginas estaticas
#			DocumentRoot /home/ricardo/servidor
#			<Directory /home/ricardo/servidor/>
#				Options Indexes FollowSymLinks Multiviews
#				AllowOverride All
#				Require all granted
#			</Directory>

			ErrorLog ${APACHE_LOG_DIR}/error.log
			CustomLog ${APACHE_LOG_DIR}/access.log combined

			Action fastcgi-script /home/ricardo/servidor/umoc_rest.fcgi

			ScriptAlias /onvif /home/ricardo/servidor/umoc_rest.fcgi

			<Directory "/home/ricardo/servidor">
				AllowOverride All
				Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
# Linha abaixo desnecessaria, porque no ficheiro fastcgi.conf tem o AddHandler definido
#				AddHandler fastcgi-script .fcgi
				Require all granted

				Script GET umoc_rest.fcgi
				Script POST umoc_rest.fcgi
				Script PUT umoc_rest.fcgi
				Script DELETE umoc_rest.fcgi
			</Directory>

# Nao e recomendado devido a desempenho e seguranca
#			AccessFileName .htaccess
#			<FilesMatch "^\.ht">
#				Require all granted
#			</FilesMatch>
		</VirtualHost>
```

Ficheiro fastcgi.conf da diretoria do apache “/mods-available” ou “/mods-enable”.

```
<IfModule mod_fastcgi.c>
  AddHandler fastcgi-script .fcgi
  #FastCgiWrapper /usr/lib/apache2/suexec
  FastCgiIpcDir /var/lib/apache2/fastcgi
  FastCgiConfig -maxProcesses 50 -maxClassProcesses 20 -startDelay 1 -killInterval 30 -idle-timeout 90
</IfModule>
```

ANEXO II – PLATAFORMA ANDROID

O desenvolvimento da aplicação para gestão de câmaras implica o estudo aprofundado sobre a plataforma Android acerca de aspetos como arquitetura, tecnologias envolvidas e componentes do sistema.

O Android é um sistema operativo *open-source* que lidera o mercado atualmente para plataformas móveis. A plataforma Android é baseada em Linux, sendo que, os fabricantes adaptam o *software* de acordo com o *hardware* dispositivo. O Android utiliza Java como linguagem de programação e XML como linguagem de estruturação da UI com regras próprias. A plataforma de desenvolvimento de aplicações recomendada pela Google, autora do Android, é o Android Studio. Primeiramente era utilizado o Eclipse para desenvolver aplicações sendo que este foi substituído pelo Android Studio. A instalação do IDE do Android Studio deve ser feita em conjunto com ao SDK (Software Development Kit) que contêm as várias APIs necessárias para desenvolvimento. O Android Studio oferece ferramentas para análise de memória, *debug* do código, criação de aplicações baseadas em exemplos geração de ficheiros executáveis apk para instalar aplicações entre outras funcionalidades. Existem inúmeras lojas de aplicações para Android sendo que a Google Play é a principal, focando-se na distribuição de aplicações e conteúdos multimédia como música e filmes. A Google Play permite aos desenvolvedores criarem uma conta que lhes para publicar aplicações *beta* ou definitivas. Em adição são disponibilizadas ferramentas de *analytics*.

Componentes da aplicação

Os componentes da aplicação [63] fazem parte da estrutura de uma aplicação Android e cada um pode ser um ponto de entrada na mesma. Nem todos os componentes são pontos de entrada e alguns dependem uns dos outros sendo que cada um deles tem uma função especifica. Existem quatro componentes distintos e cada um tem um ciclo de vida que define como o componente é criado e destruído. Os componentes existentes são:

- **Activities:** uma *activity* representa um ecrã único na interface de utilizador que é personalizado para interagir com o utilizador. As *activities* trabalham em conjunto para proporcionar ao utilizador uma experiencia agradável de interação com o ecrã. As *activities* são independentes umas das outras e podem existir várias criadas ao mesmo tempo. Estas ficam guardadas numa *stack* (denominada "*back stack*") onde se insere uma *activity* anterior sempre que se inicia uma nova *activity,* sendo que as anteriores ficam em pausa. Uma *activity* é

implementada como uma subclasse da Activity. A *activity* implementa *callbacks* que fazem parte do seu ciclo de vida. O ciclo de vida de uma *activity* é descrito na Figura II.0.1 [64] que representa a transição de estado através das *callbacks*. Quando uma *activity* é iniciada é executado o onCreate() e quando termina é executado o onDestroy(). Existem ainda *callbacks* para o onStart(), onRestart(), onResume(), onPause() e onStop().

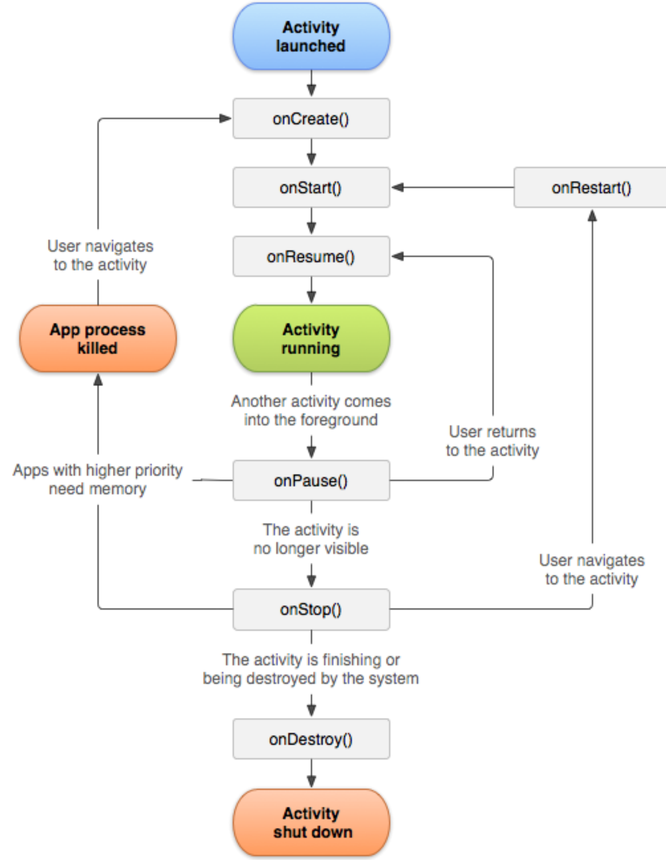


Figura II.0.1 – Ciclo de vida da Activity [64]

- **Services:** um *service* [65] é uma componente que corre em *background* para fazer operações de longa execução ou trabalho de processos remotos. Um serviço não tem interface de utilizador tal como uma *activity* e não precisa de interagir com o utilizador. Por exemplo, um serviço pode fazer operações enquanto o utilizador está em aplicações diferentes, tais como reproduzir música ou recolher dados da rede sem bloquear a interação com uma *activity.* Um serviço pode ser iniciado por outro componente como uma *activity,* o qual, pode ficar a correr ou ligar para ele com a finalidade de interagir mutuamente. O serviço é implementado pela subclasse Service. Assim, deve-se usar um serviço em detrimento de uma *thread* caso seja necessário realizar operações longas que não interajam diretamente com o utilizador. Os serviços podem assumir a forma Started ou Bound. Um serviço está na forma "*started*"

quando um componente da aplicação invoca o startService(). Posteriormente o serviço corre em *background* indefinidamente mesmo que o componente que o iniciou seja destruído. Normalmente um serviço não devolve valores ao componente que o invoca, por exemplo, no processo de *download* de um ficheiro através da rede o serviço para automaticamente. Um serviço está a "*bound*" quando uma componente da aplicação invoca o bindService(). Este tipo de serviço oferece uma interface cliente-servidor que permite aos componentes interagirem com o serviço, enviar, receber, obter resultado e permite fazê-lo através de *interprocess communication* (IPC). Neste caso o serviço corre enquanto o componente da aplicação estiver ligado a ele. Múltiplos componentes podem-se ligar ao serviço, mas quando todos se desligam, o serviço é destruído.

- **Content providers:** um *content provider* [66] gere um conjunto partilhado de dados que pode ser armazenado no sistema de ficheiros, bases de dados SQLite, na *cloud* ou um provedor de serviços caso o conteúdo seja permitido. Os *content providers* podem ser usados para modificar dados através de pedidos se o mesmo o permitir. Também são úteis para ler e escrever dados privados e que não são partilhados. Um *content provider* deve implementar a subclasse ContentProvider e deve implementar um conjunto padrão de APIs que permitem executar transações a outras aplicações.
- **Broadcast receivers:** um *broadcast receiver* [67] é um componente que responde às mensagens transmitidas de todo o sistema. Caso não seja necessário transmitir mensagens entre aplicações, recomenda-se o uso do LocalBroadcastManager, em alternativa a outras classes de *broadcats.* Há duas principais classes para receber as mensagens transmitidas, "Normal" e "Ordered" *broadcasts*. Para o "Normal broadcast" envia-se com "Context.sendBroadcast" que é totalmente assíncrono. Neste caso, todos os recetores de transmissões são executados numa ordem indefinida, e muitas vezes, ao mesmo tempo. Esta opção é mais eficiente, sendo que, implica que os recetores não utilizem o resultado. Para o "Ordered broadcast" envia-se com "Context.sendOrderedBroadcast" que são entregues a um recetor de cada vez. Assim, à medida que cada recetor acaba a sua execução, pode propagar o resultado para o próximo recetor ou pode parar o *broadcast* para que não seja passado para os próximos recetores.

Há componentes que auxiliam na utilização dos 4 componentes principais. Estes componentes são usados em conjunto, permitindo, desenhar e definir o modo de interação com o ecrã. Os principais componentes auxiliares são:

- **Fragments:** um fragmento [57] ajuda a representar partes da interface de utilizador e pode influenciar o comportamento da mesma numa *activity*. É possível a utilização de múltiplos fragmentos numa *activity,* inclusive, a sua reutilização com a finalidade de construir uma interface de utilizador com vários painéis e com determinados comportamentos. Um fragmento faz parte de uma *activity* e tem um ciclo de vida distinto que está representado na Figura II.0.2. O fragmento tem *callbacks* que podem ser usadas para influenciar o seu comportamento, tais como, definir o comportamento na sua criação ou quando é destruído. Existem subclasses de fragmento em alternativa ao Fragment com funcionalidades e comportamentos implementados, por exemplo, ListFragment que exibe uma lista de itens que são geridos por um *adapter* semelhante à ListActivity*.*

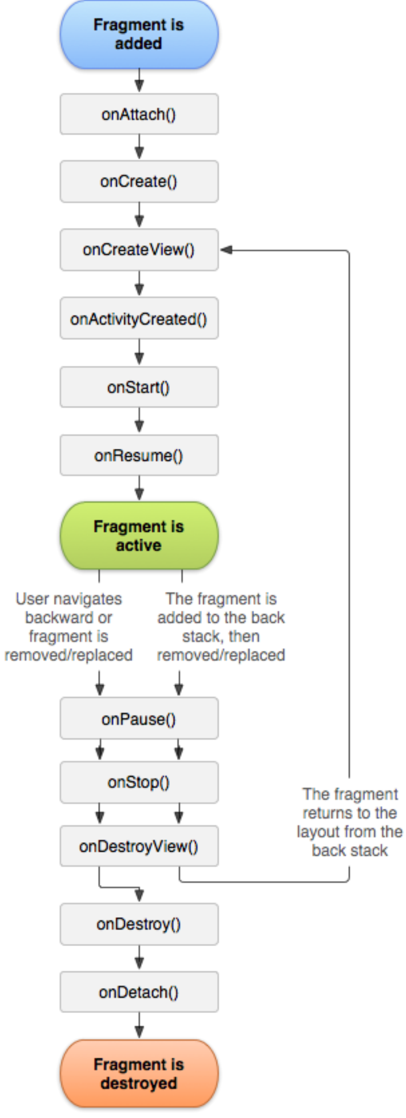


Figura II.0.2 - Ciclo de vida de um Fragment enquanto a sua *activity* está aberta [57]

- **Views:** as Views [68] usadas através da classe View que é um bloco básico para construção os componentes da interface de utilizador. Esta representa uma área retangular no ecrã e é responsável por tratar de eventos e pelo desenho da interface. Os *Widgets* da interface de utilizador recorrem à View para fazer componentes de interação como botões e campos textuais, imagens, listas entre outros. As Views têm um conjunto de propriedades que as definem, tais como, tamanho de texto, *padding,* etc. As propriedades podem ser definidas através de métodos e ficheiros XML. Além disso, é possível configurar *listeners* para definir comportamentos da View.
- **Layouts:** os layouts [69] definem uma estrutura visual da UI para uma *activity* ou *widget* da aplicação*.* Os elementos de um *layout* podem ser declarados em ficheiros XML em que o vocabulário corresponde às classes e subclasses de uma View. Em adição, é possível instanciar os elementos em tempo de execução recorrendo a objetos de View ou ViewGroup programaticamente que permite manipular as suas propriedades.
- **Intents:** uma Intent [70] é uma descrição abstrata de operações a executar. Tipicamente é usada quando se inicia uma *activity* através do startActivity e pode ser usada com um BroadcastReceiver quando se inicia um serviço ou se conecta um serviço através dos métodos definidos para os mesmos. O uso mais comum é a passagem de dados estruturados através de *activities.*
- **Resources:** os recursos [71] como dimensões, *strings*, subpartes de *layouts* devem ser sempre exteriorizados do código da aplicação de modo a fazer uma manutenção dos mesmos independente. Este fator permite disponibilizar funcionalidades como o suporte para diferentes línguas, tamanhos de ecrã e mesmo escolher imagens consoante o tamanho do ecrã. Para suportar diferentes configurações de acordo com as características do dispositivo os recursos são organizados na diretoria "res/" do projeto da aplicação que usa subdiretorias de acordo com o tipo e configuração. Existem recursos por defeitos que são usados independentemente do dispositivo e recursos alternativos para serem utilizados por dispositivos com características específicas.
- **Manifest**: o ficheiro **Manifest** (**AndroidManifest.xml**) [72] existe em todas as aplicações para fazer configurações essenciais ao funcionamento da aplicação. Todas as permissões necessárias devem ser declaradas neste ficheiro, assim como, todas as atividades existentes para utilização, versões da API, serviços, *content providers*, nome da aplicação, temas entre outros.

ANEXO III – FUNÇÕES SUPORTADAS PELO SERVIÇO WEB REST

Neste anexo é descrita as funções suportadas pela aplicação servidora com recurso a diagramas com a descrição da URI e tipo/nome do método HTTP. Os serviços têm uma função base que dá acesso às URIs disponíveis para os respetivos serviços.

A função getServiceCapabilities do serviço Device Managment permite obter as *capabilities* de um dispositivo em específico. É das funções mais importantes, uma vez que, permite verificar quais os serviços suportados e as capacidades do dispositivo. Pode-se obter a informação de um dispositivo através da função de GetDeviceInformation e existem funções para editar/obter a data e a hora, configurações DNS e DNS dinâmico e configurações do Network Time Protocol (NTP) de um dispositivo. Pode-se consultar/editar/eliminar os parâmetros de escopo. Existem ainda as 4 funções de CRUD sobre os utilizadores. Relativamente às especificações ONVIF atuais, para o serviço Device Managment, faltam implementar as funcionalidades de Network (operações de *hostname*), System (*Backup*, *restore* e *firmware*) e Security (operações sobre políticas e certificados).

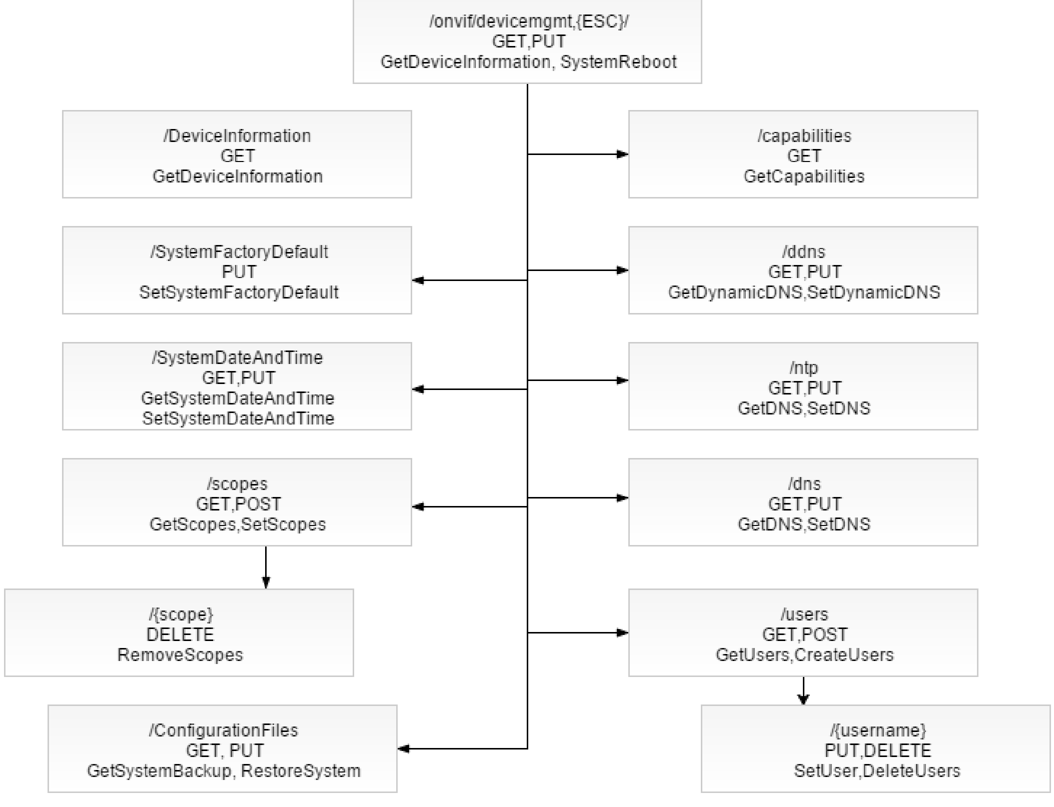


Figura III.0.1 - Diagrama do serviço Device Managment

No serviço de Media a função GetProfiles devolve as funções para criar, editar e eliminar perfis, pode-se consultar a URI de *Stream*, *Snapshot,* obter *codecs* e a lista de *video sources* e

começar/terminar uma *stream.* A função de obter os perfis de *media* é a que tem maior volume de dados. Relativamente às especificações ONVIF atuais, para o serviço Media faltam operações para obter entidades (*audio sources, audio decoder, video encoder, video analytics, video source, metadata, audio encoder* e *audio output*) compatíveis com determinado perfil de media.

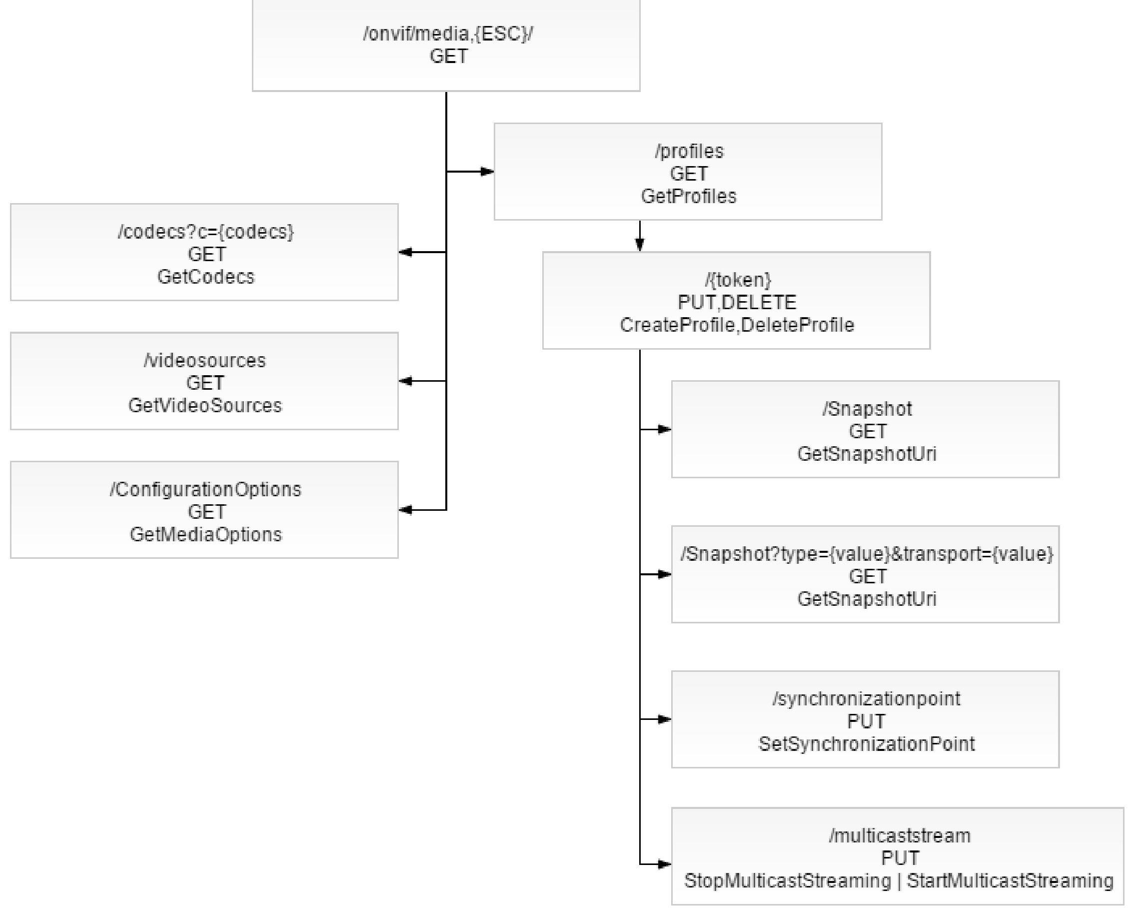


Figura III.0.2 - Diagrama do serviço Media

No serviço Device IO estão desenvolvidas funções para obter e alterar configurações, estado e consultar as opções disponíveis das saídas físicas de um dispositivo. Há um método para consultar as entradas físicas. Do mesmo modo são disponibilizados métodos para obter as portas série e enviar e receber dados. Há métodos para obter as opções e configurações para a porta série assim como alterar as suas configurações. Relativamente às especificações ONVIF atuais, para o serviço Device IO faltam implementar operações para *relay outputs*, *serial ports* e *digital inputs.*

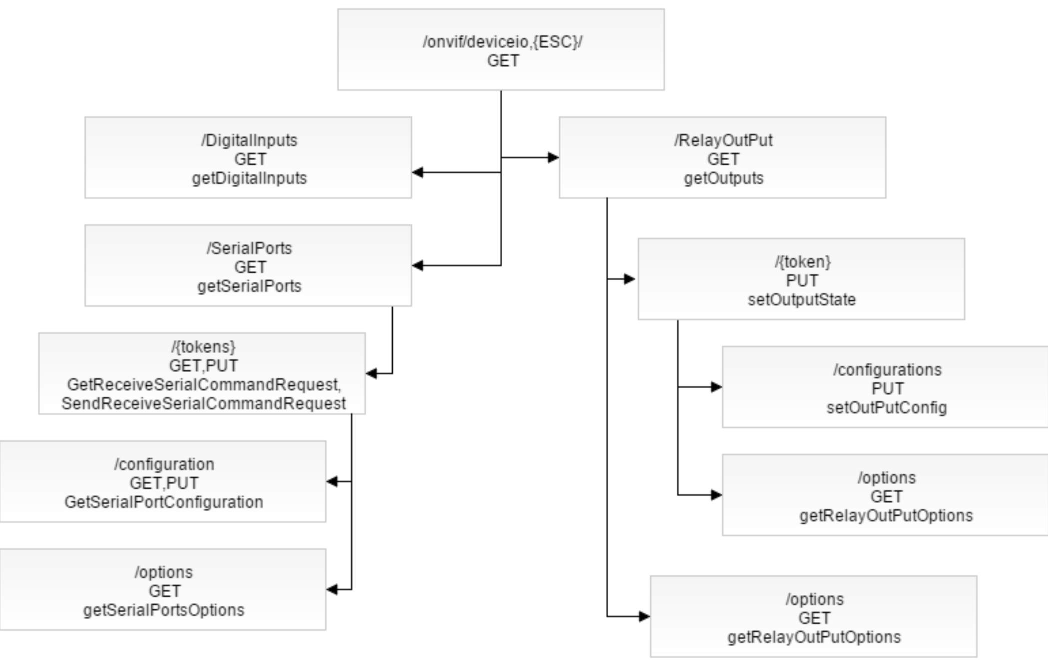


Figura III.0.3 - Diagrama do serviço DeviceIO

O serviço Imaging tem métodos que permitem consultar e alterar as configurações de imagem de um dado *video source.* As configurações de imagens são o brilho, contraste, *sharpness*, saturação, focagem entre outras. Pode-se consultar o estado atual das propriedades da imagem, consultar as opções de movimento de lente e definir o movimento de lente de um *video source* ou tipo de focagem de lente. Relativamente às especificações ONVIF atuais, para o serviço Imaging que é bastante completo a nível de funcionalidades falta operações para os *presets.*

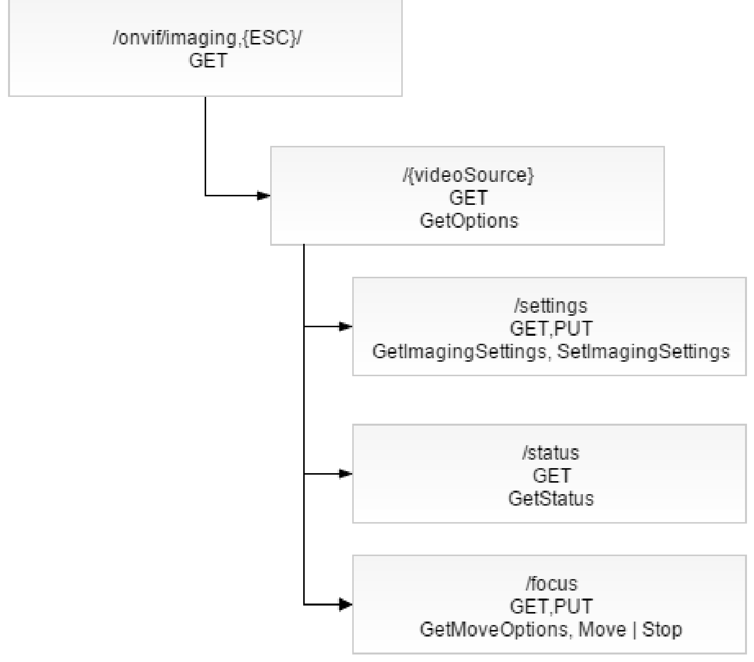


Figura III.0.4 - Diagrama do serviço Imaging

Para o serviço PTZ pode-se fazer movimentos de *pan, tilt* e *zoom* dado um perfil media. Os movimentos podem ser do tipo relativo, absoluto ou contínuo. Outras funcionalidades são consultar, alterar, criar e eliminar *presets* assim como alterar ou consultar a *home position* de um perfil de *media*. Relativamente às especificações ONVIF atuais, para o serviço PTZ faltam operações para os *nodes*, *configurations* e *configurations options*.

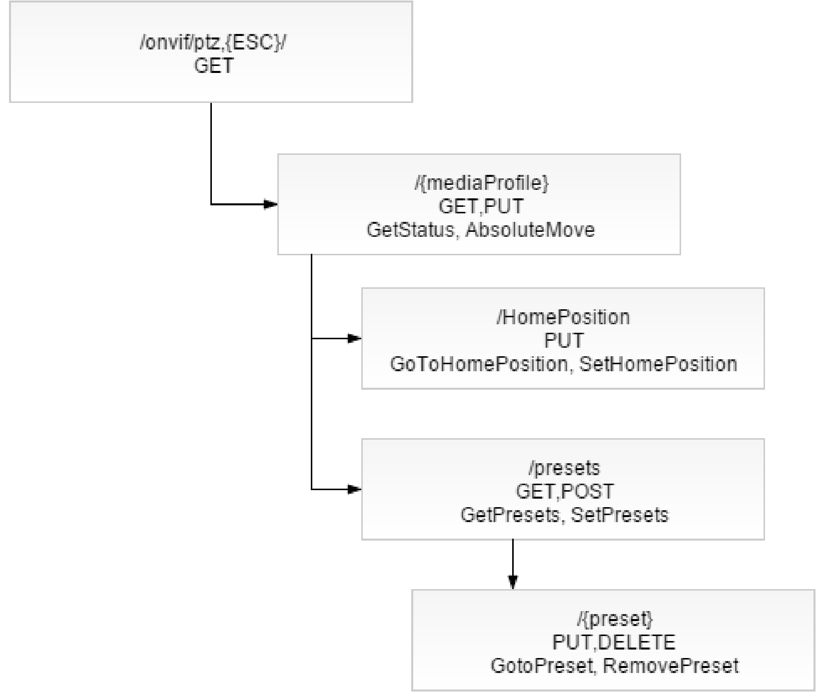


Figura III.0.5 - Diagrama do serviço PTZ

No serviço Event é possível consultar os eventos ocorridos num dispositivo específico, criar/terminar eventos e renovar uma subscrição de eventos que esteja ativa. A duração de uma subscrição de eventos pode ser renovada ou terminada.

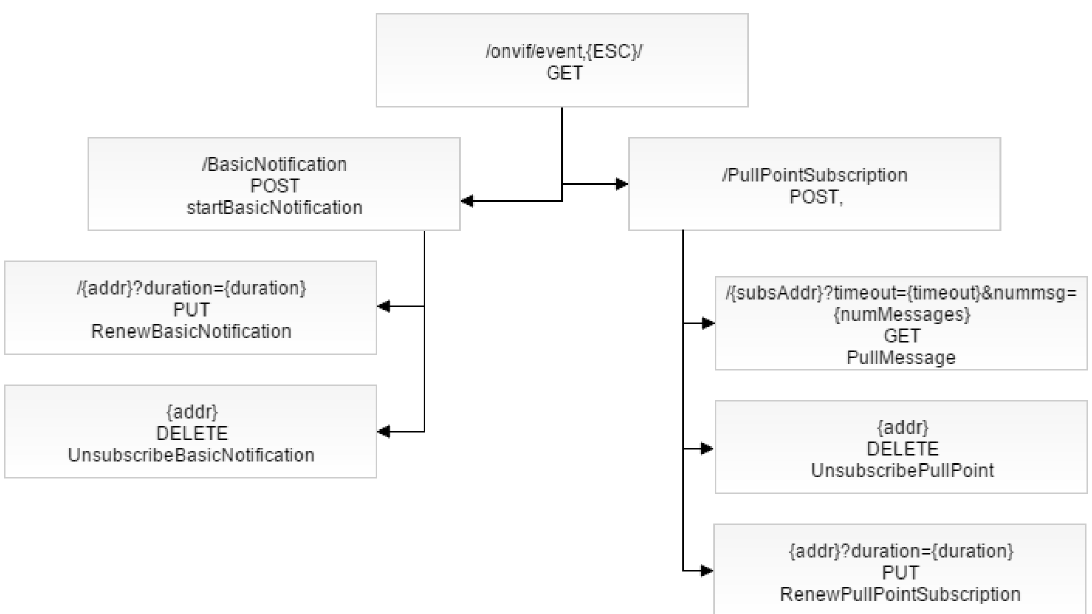


Figura III.0.6 - Diagrama do serviço Events

ANEXO IV – FUNDAMENTOS JSON

Neste capitulo é feita uma breve descrição dos fundamentos de JSON. Em JSON, a definição de objeto é um conjunto não ordenado de pares nome/valor [73]. Começa por "{" e termina com "}". Cada nome é separado por ":" e os pares nome/valor são separados por ",". O formato de um objeto JSON está representada na Figura IV.0.1.

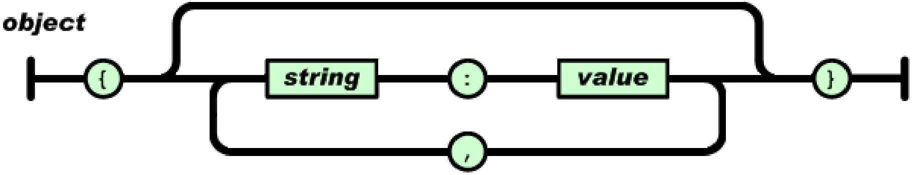


Figura IV.0.1 - Formato de um *object* JSON [73]

Um *array* é uma coleção ordenada de valores. Utiliza "[" e "]" para começar e terminar respetivamente e os valores são separados por ",". O formato de um *array* é representado na Figura IV.0.2.

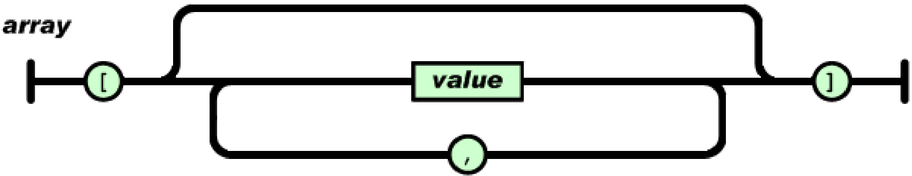


Figura IV.0.2 - Formato de um *array* JSON [73]

Um valor pode ser uma *string,* número, objeto, *array,* valor booleano ou *null.* O formato do valor é representado na Figura IV.0.3.

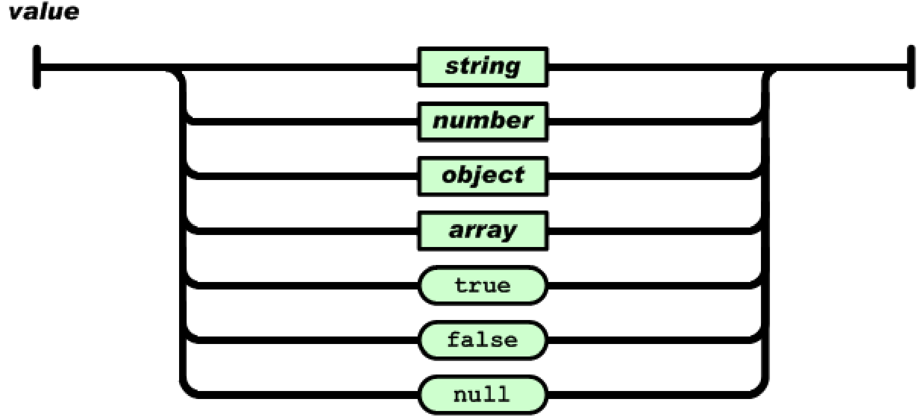


Figura IV.0.3 - Formato do *value* JSON [73]

ANEXO V – CLASSES DE DADOS

Neste anexo é feita uma descrição das classes de dados desenvolvidas e apresentado os diagramas de classes genéricos das mesmas. Dividiu-se algumas classes de dados por blocos, uma vez que, a quantidade de classes é significativa, conseguindo assim uma melhor leitura dos diagramas. Os diagramas apresentados localizam-se no *package* respetivo ao serviço referido, p. ex., para o serviço DeviceIO associa-se o *package* “org.uminho.onvif.deviceio”.

Os diagramas de classes de dados não apresentam campos nem construtores/métodos devido à quantidade significativa de classes e de informações.

Device IO

O diagrama de classes de dados da Figura V.0.1 para o serviço DeviceIO tem as classes de dados deste serviço. A classe principal é OutputInformation que guarda informações como o *token*, *delaytime*, RelayIdleState com os valores “Closed” ou “Open” e RelayMode com os valores “Monostable” e “Bistable”. Para guardar as *capabilities* deste serviço usa-se a classe DeviceIoCapabilities que contém o número de fontes/saídas de vídeo/áudio, portas série, entradas digitais e saídas de retransmissão.

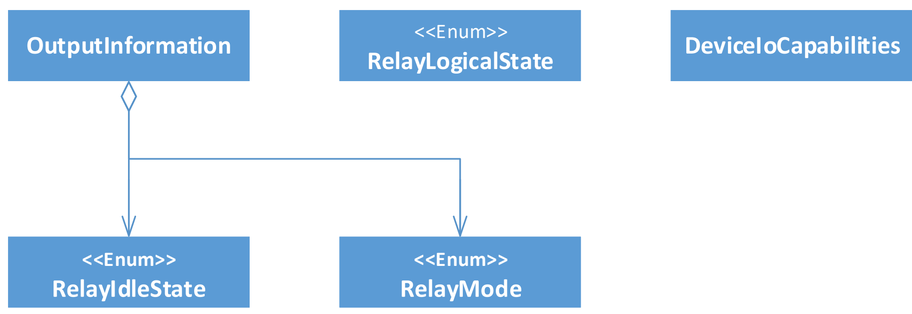


Figura V.0.1 - Diagrama de classes de dados do serviço DeviceIO

Device Managment

As *capabilities* do serviço Device Managment são guardadas na classe DeviceCapabilities da Figura V.0.2 e tem uma subparte de *capabilities* divididas em System, Miscellaneous, Security e Network. Estas classes contêm diversos campos na maioria do tipo booleanos sobre as *capabilities* do Device.

Figura V.0.2 - Diagrama de classes de dados para as capabilities do DeviceMgmt

O diagrama da Figura V.0.3 tem as classes de dados para informações de conta de utilizador que guarda o nome e nível de acesso.

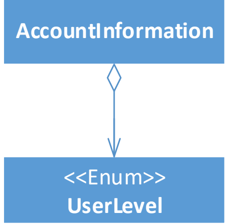


Figura V.0.3 - Diagrama de classes de dados para a AccountInformation do Device Managment

O diagrama da Figura V.0.4 tem as classes de dados para guardar as informações dinâmicas de DNS como o tipo, nome e TTL.

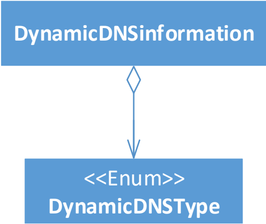


Figura V.0.4 - Diagrama de classes de dados para a DynamicDNSInformation do Device Managment

O diagrama da Figura V.0.5 tem as classes de dados para guardar informações de data e hora. As classes de dados têm informações de como é obtida a data e hora, ou seja, NTP ou manualmente, fuso horário, UTC e local, *time zone* e se se usa *day light saving*.

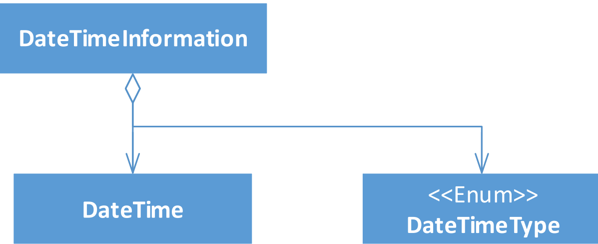


Figura V.0.5 - Diagrama de classes de dados para a DateTimeInformation do Device Managment

Para os *scopes* há duas classes para guardar o tipo e designação do *scope*. O diagrama de classes dos scopes pode ser consultado na Figura V.0.6.

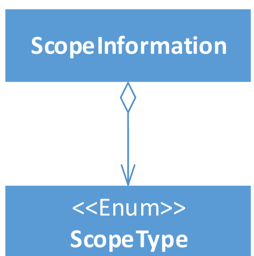


Figura V.0.6 . Diagrama de classes de dados para a ScopeInformation do Device Managment

As classes para DNS têm parâmetros para especificar se o DNS é manual ou por DHCP, domínio de procura, IP e tipo de IP. O diagrama de classes de DNS pode ser consultado na Figura V.0.7.

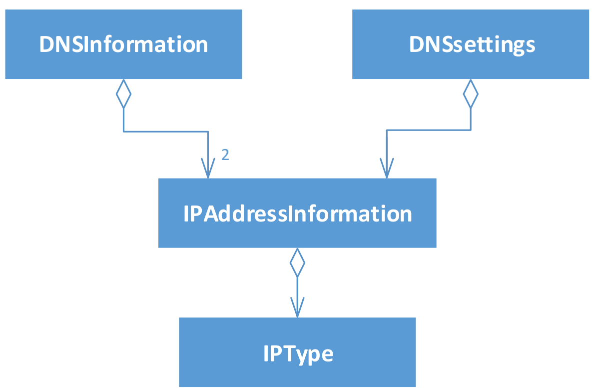


Figura V.0.7 - Diagrama de classes de dados para a DNSInformation e DNSSettings do Device Managment

O diagrama de classes para *factory type*, informação de dispositivo e tipo de serviço pode ser consultado na Figura V.0.8.

Figura V.0.8 - Diagrama de classes de dados para a DeviceInformation, Service e FactoryDefaultType do Device Managment

As classes para NTP têm parâmetros para especificar se o NTP é manual ou por DHCP, endereço e tipo de *host* (IPv4, IPv6, DNS, DHCP). O diagrama de classes para NTP pode ser consultado na Figura V.0.9.

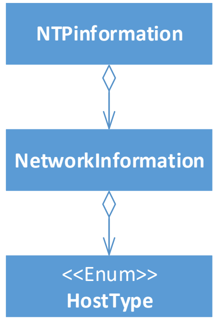


Figura V.0.9 - Diagrama de classes de dados para a NTPInformation do Device Managment

As classes para os serviços NVT têm parâmetros para especificar quais os serviços de um NVT, endereço de serviço e a versão ONVIF. O diagrama de classes pode ser consultado na Figura V.0.10.

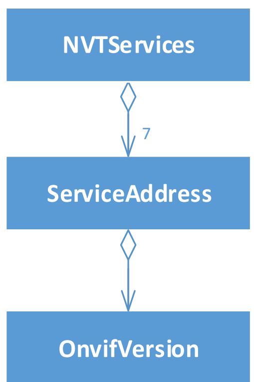


Figura V.0.10 - Diagrama de classes de dados para a NVTServices do Device Managment

Discovery

. O diagrama de classes para os *scopes* de um NVT e para informação de um dispositivo pode ser consultado na Figura V.0.11. As informações de um dispositivo consistem na descrição do tipo de código, localizações, *hardware*, nomes, *others* e endereço. Os tipos de dispositivos estão divididos em NVT, NVD, NVS, NVA e *any*. A classe NvtScope é um enumerado para a forma de uso dos scopes do NVT.

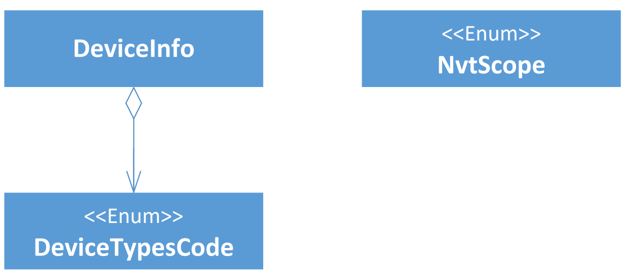


Figura V.0.11 - Diagrama de classes de dados de dados do serviço Discovery

Events

O diagrama de classes de dados para o serviço Events pode ser consultado na Figura V.0.12. As classes guardam uma lista de mensagens de notificações com tópico e mensagem, o endereço e tempo de fim para uma subscrição, endereço, tempo e número de mensagens para um pedido de mensagem, e as *capabilities* do serviço que têm parâmetros opcionais para funcionalidades do serviço.

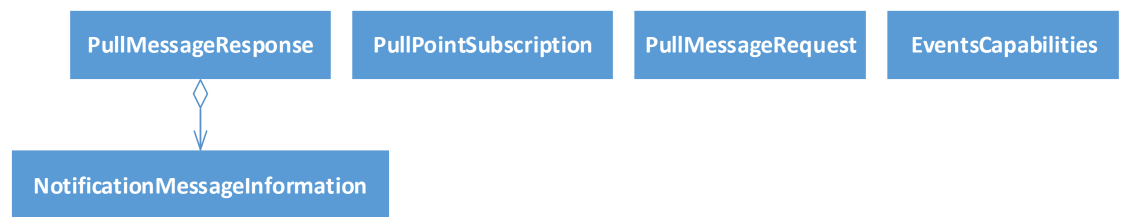


Figura V.0.12 - Diagrama de classes de dados do serviço Events

Imaging

O diagrama de classes das *capabilities* de Imaging pode ser consultado na Figura V.0.13. As *capabilities* consistem num parâmetro opcional para a estabilização de imagem.

ImagingCapabilities

Figura V.0.13 - Diagrama de classes de dados para as *capabilities* do serviço Imaging

O diagrama de classes de dados para os tipos de focagem pode ser consultado na Figura V.0.14. Os tipos de focagem podem ser relativos, absolutos ou contínuos. O absoluto tem os parâmetros velocidade e posição, o relativo tem distância e velocidade e o contínuo tem apenas velocidade.

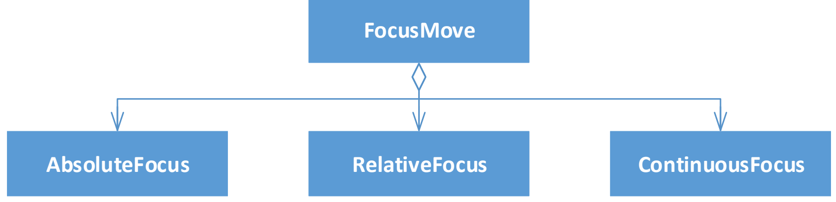


Figura V.0.14 - Diagrama de classes de dados para os movimentos de *focus* do serviço Imaging

O diagrama de classes de dados para as opções de movimento pode ser consultado na Figura V.0.15. Existem opções para os tipos de movimento absoluto, relativo e contínuo cujos parâmetros são opcionais. Para o movimento contínuo para além de opcional tem uma gama de valores para o parâmetro de velocidade.

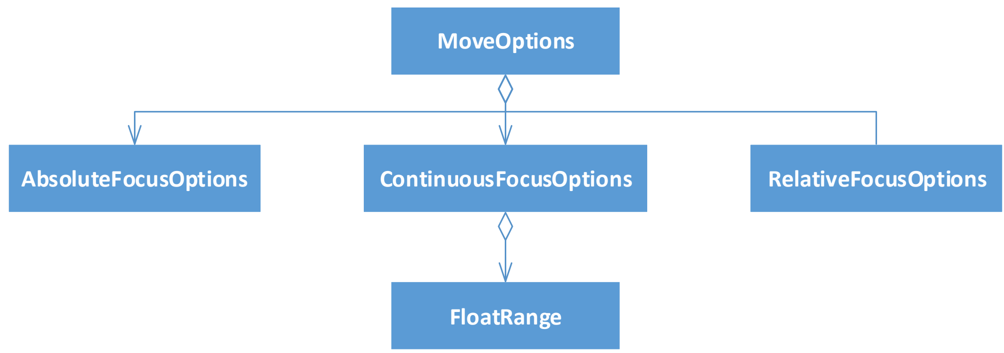


Figura V.0.15 - Diagrama de classes de dados para as opções de movimentos do serviço Imaging

O diagrama de classes de dados para o estado de imagem pode ser consultado na Figura V.0.16. As informações de estado são a posição, estado de movimento e erro. O estado de movimento pode ser desconhecido, a mover-se ou inativo.

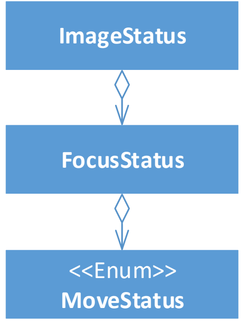


Figura V.0.16 - Diagrama de classes de dados para o estado de imagem do serviço Imaging

Os diagramas de classes de dados para as configurações de imagem foram divididos em 6 diagramas por níveis hierárquicos que se completam. Desta forma a visualização do diagrama é facilitada.

O diagrama de classes de dados para as configurações de imagem número 1 de 6 pode ser consultado na Figura V.0.17. As definições de imagem estão divididas em *white balance information, wide dynamic range information, ircut filter mode, focus configuration, back light compensation* e *exposure settings.* Existe ainda definições mais gerais para *brightness, color saturation, contrast* e *sharpness* na classe ImageSettings.

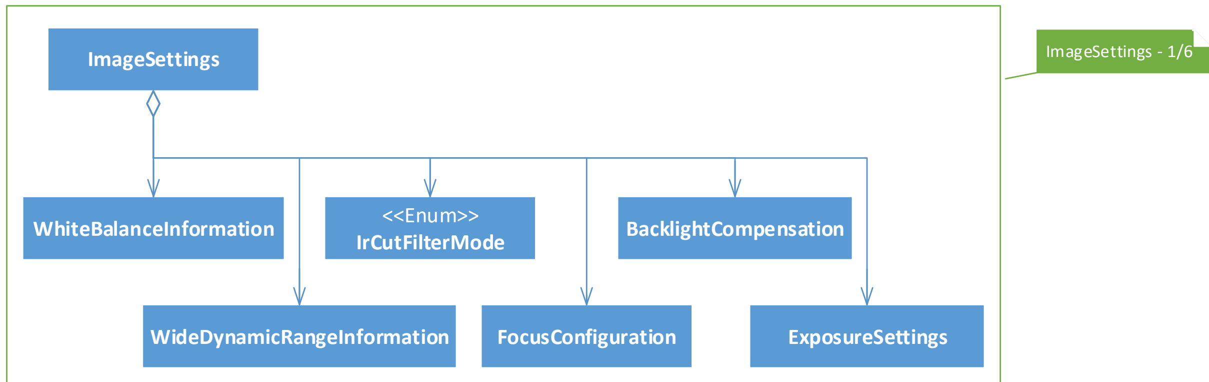


Figura V.0.17 - Diagrama de classes de dados das definições de imagem para o serviço Imaging

O diagrama de classes de dados número 2 de 6 para *white balance information* pode ser consultado na Figura V.0.18. As classes do diagrama têm informações do modo (MANUAL/AUTO), *cr gain e cb gain*.

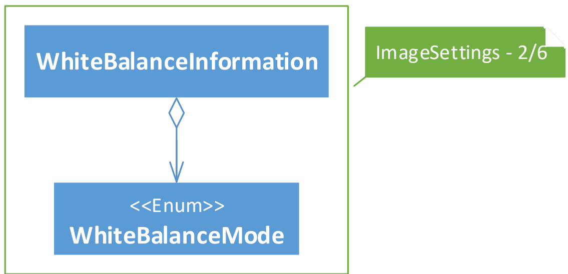


Figura V.0.18 - Diagrama de classes de dados para *white balance information* do serviço Imaging

O diagrama de classes de dados número 3 de 6 para *wide dynamic range information* pode ser consultado na Figura V.0.19. As classes do diagrama têm parâmetros para o nível e modo (ON/OFF).

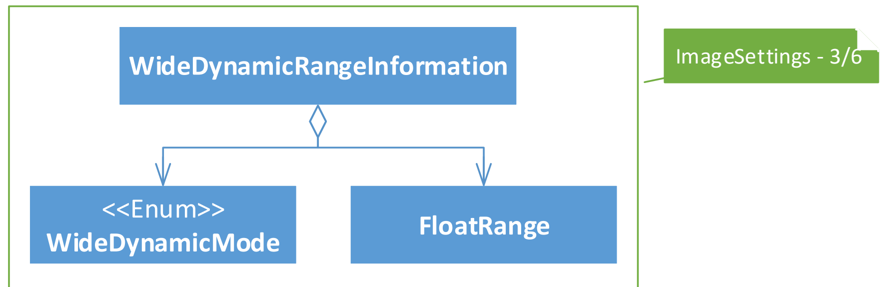


Figura V.0.19 - Diagrama de classes de dados para *wide dynamic range information* do serviço Imaging

O diagrama de classes de dados número 4 de 6 para *focus configuration* pode ser consultado na Figura V.0.20. As classes do diagrama têm parâmetros para a *default speed, near limit, far limit* e *auto focus mode* (MANUAL/AUTO).

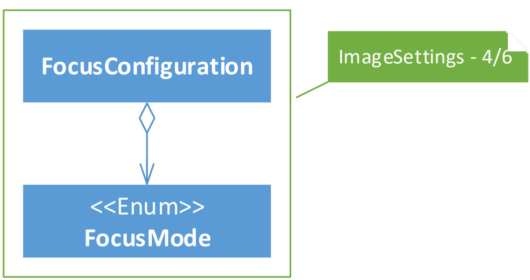


Figura V.0.20 - Diagrama de classes de dados para *focus configuration* do serviço Imaging

O diagrama de classes de dados número 5 de 6 para *back light compensation* pode ser consultado na Figura V.0.21. As classes do diagrama têm parâmetros para o modo (ON/OFF) e nível.

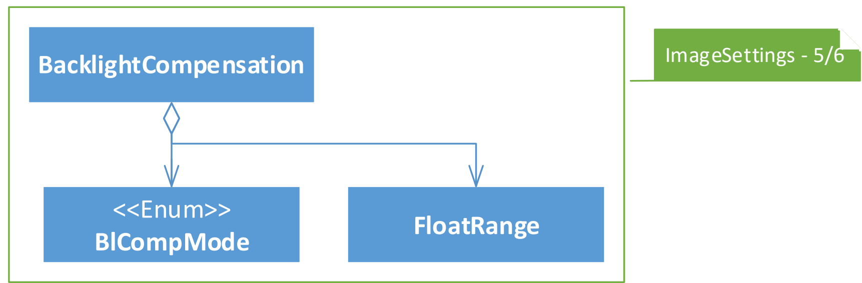


Figura V.0.21 - Diagrama de classes de dados para *back light compensation* do serviço Imaging

O diagrama de classes de dados número 6 de 6 para *exposure settings* pode ser consultado na Figura V.0.22. As classes do diagrama têm parâmetros *para maximum exposure time, maximum gain, maximum iris, exposure time, gain, iris, minimum exposure time, minimum gain, minimum iris*, *mode* (MANUAL/AUTO), *priority* (LowNoise/FrameRate) e *window* (*bottom, top, left* e *right*).

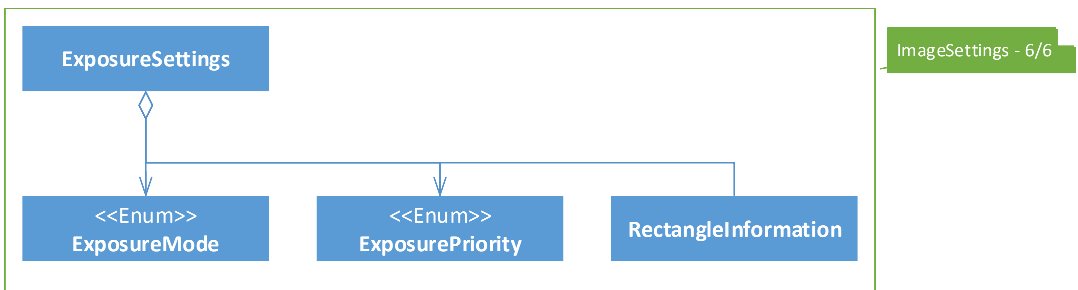


Figura V.0.22 - Diagrama de classes de dados para *exposure settings* do serviço Imaging

Os diagramas de classes de dados para as opções de imagem foram divididos em 6 diagramas por níveis hierárquicos que se completam para facilitar a sua visualização. Os parâmetros das opções consistem em listas de modos disponíveis, valores possíveis e intervalos de valores em conformidade com as configurações de imagem.

O diagrama de classes de dados número 1 de 6 para as opções de imagem pode ser consultado na Figura V.0.23. As opções de imagem estão divididas em lista do enumerado IrCutFilterMode, *back light compensation options, exposure options, wide dynamic range options* e

white balance options. Existem ainda opções gerais para *brightness, color saturation, contrast* e *sharpness* do tipo FloatRange.

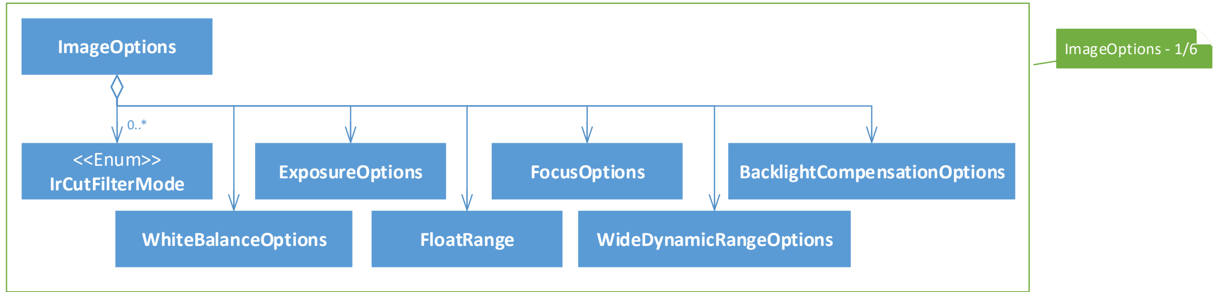


Figura V.0.23 - Diagrama de classes de dados para *image options* do serviço Imaging

O diagrama de classes de dados número 2 de 6 para as *white balance options* pode ser consultado na Figura V.0.24. As classes deste diagrama têm parâmetros para *yr gain, yb gain* e lista de modos disponíveis.

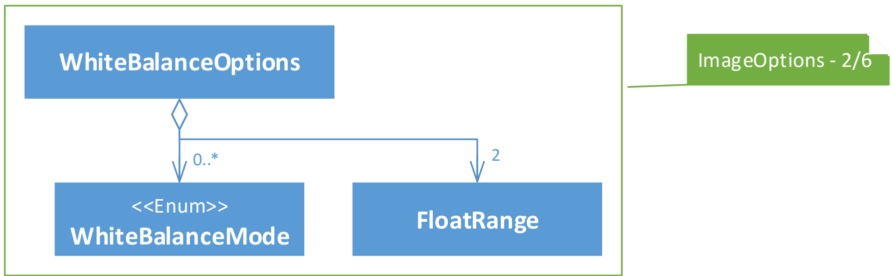


Figura V.0.24 - Diagrama de classes de dados para *white balance options* do serviço Imaging

O diagrama de classes de dados número 3 de 6 para as *exposure options* pode ser consultado na Figura V.0.25. As classes deste diagrama têm parâmetros para lista de modos e intervalo de valores para os parâmetros do diagrama Figura V.0.22.

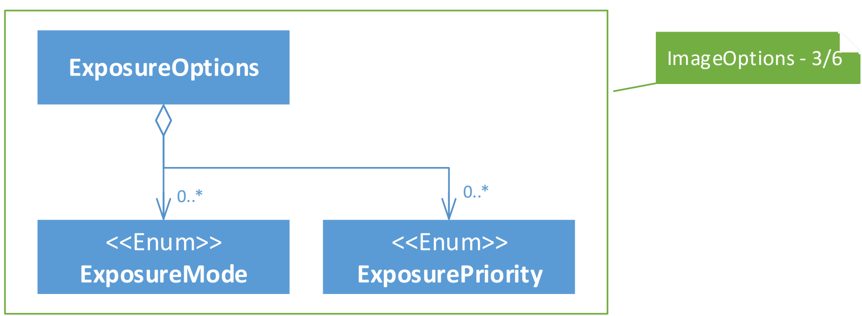


Figura V.0.25 - Diagrama de classes de dados para *exposure options* do serviço Imging

O diagrama de classes de dados número 4 de 6 para as *focus options* pode ser consultado na Figura V.0.26. As classes deste diagrama têm parâmetros para a lista de modos e intervalo de valores para os parâmetros do diagrama Figura V.0.20.

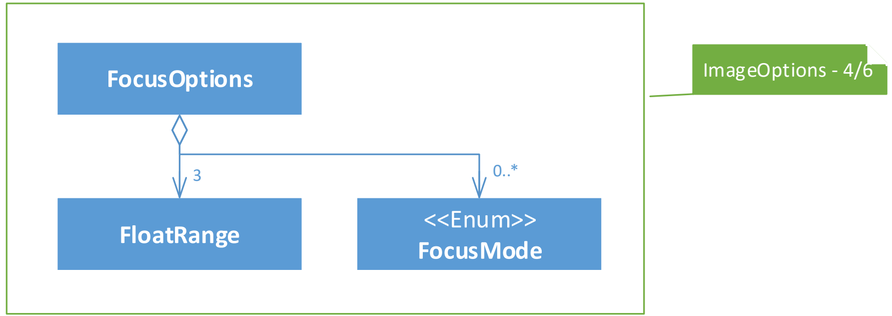


Figura V.0.26 - Diagrama de classes de dados para *focus options* do serviço Imging

O diagrama de classes de dados número 5 de 6 para as *wide dynamic range options* pode ser consultado na Figura V.0.27. As classes deste diagrama têm parâmetros para a lista de modos e intervalo de valores para os parâmetros do diagrama Figura V.0.19.

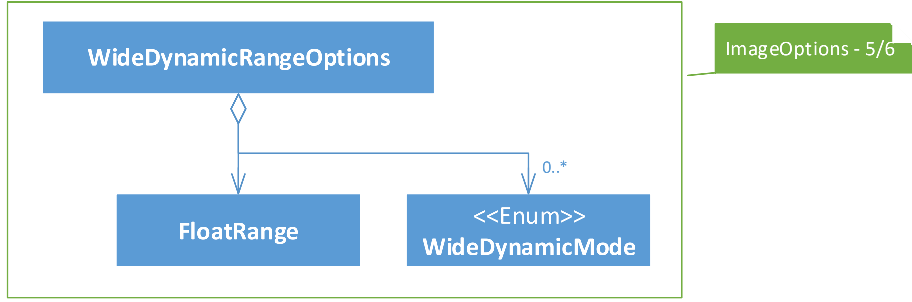


Figura V.0.27 - Diagrama de classes de dados para *wide dynamic range options* do serviço Imaging

O diagrama de classes de dados número 6 de 6 para as *back light compensation options* pode ser consultado na Figura V.0.28. As classes deste diagrama têm parâmetros para a lista de modos e intervalo de valores para os parâmetros do diagrama Figura V.0.21.

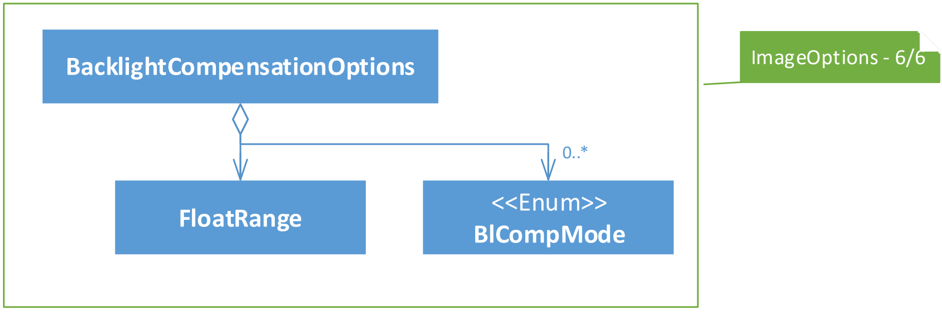


Figura V.0.28 - Diagrama de classes de dados para *back light compensation options* do serviço Imaging

Media

O diagrama de classes das *capabilities* de Media pode ser consultado na Figura V.0.29. As *capabilities* consistem em parâmetros de valor opcional para o número máximo de perfis, rotação, URI do *snapshot* e *streaming*. O parâmetro de *streaming* especifica ainda os parâmetros rtpMulticast, rtp_TCP, rtp_TSP_TCP e nonAggregateControl.

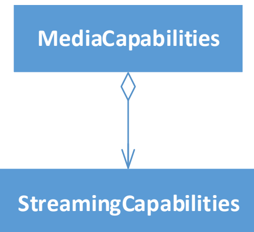


Figura V.0.29 - Diagrama de classes de dados para *media capabilities* do serviço Media

O diagrama de classes para *codecs* pode ser consultado na Figura V.0.30. Os parâmetros das classes consistem numa lista de resoluções (altura, largura e URIs). Existem URIs para *snapshot, unicast* UDP*, unicast* TCP*, unicast* RTSP*, unicast* HTTP*, multicast* UDP*, multicast* TCP*, multicast* RTSP *e multicast* HTTP*.* Existe ainda um enumerado para as opções de *codecs* (JPEG/MPEG4/H264)*.*

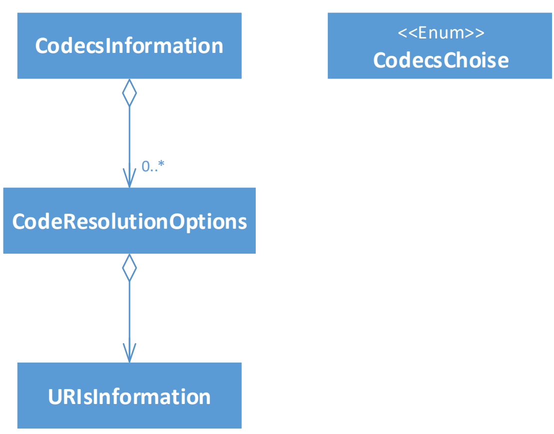


Figura V.0.30 - Diagrama de classes de dados para *codecs* do serviço Media

O diagrama de classes de dados das opções de menu foi dividido em 10 partes devido a sua complexidade de modo a facilitar a sua visualização. O diagrama parte 1 de 10 pode ser consultado na Figura V.0.31. As opções de menu têm parâmetros para as *video encoder configuration options, video source configuration options, audio source configuration options, audio encoder configuration options, metadara configuration options* e PTZ *configuration options*.

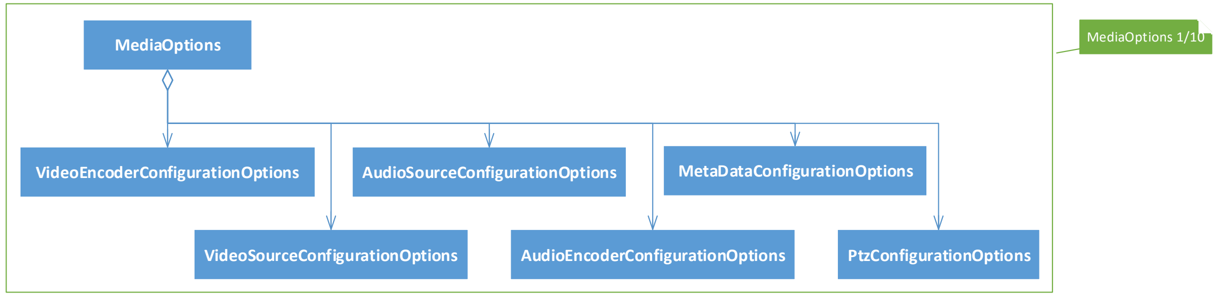


Figura V.0.31 - Diagrama de classes de dados para *media options* do serviço Media

O diagrama de classes de dados das *media options* parte 2 de 10 para as *video encoder configuration options* pode ser consultado na Figura V.0.32. Este diagrama tem parâmetros para a qualidade de alcance (mínimo e máximo) e opções para JPEG/MPEG4/H264.

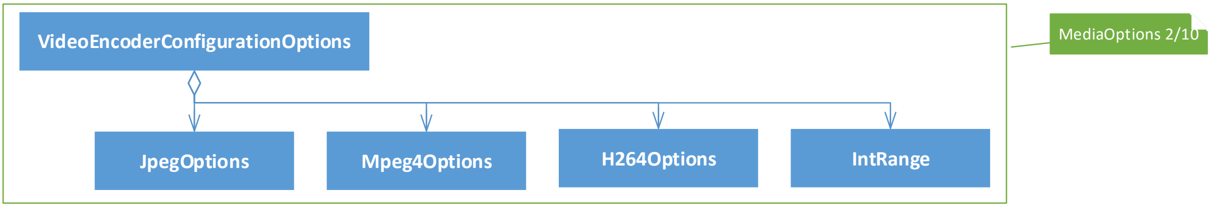


Figura V.0.32 - Diagrama de classes de dados para *video encoder configuration options* do serviço Media

O diagrama de classes de dados das *media options* parte 3 de 10 para as JPEG *options* pode ser consultado na Figura V.0.33. Neste diagrama existem parâmetros para a lista de resoluções (altura e largura), *frame rate* (máximo e mínimo) e *encoding interval* (máximo e mínimo).

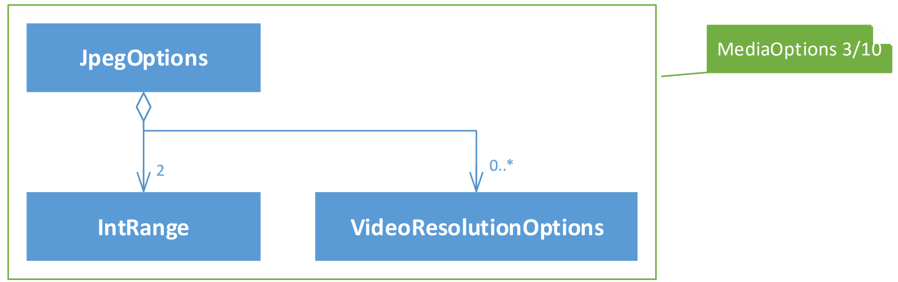


Figura V.0.33 - Diagrama de classes de dados para JPEG *options* do serviço Media

O diagrama de classes de dados das *media options* parte 4 de 10 para as MPEG4 *options* pode ser consultado na Figura V.0.34. Neste diagrama existem parâmetros para a lista de resoluções

(altura e largura), *frame rate* (máximo e mínimo), *encoding interval* (máximo e mínimo), GOV *length range* (máximo e mínimo) e lista dos perfis de MPEG4 (podem ser SP ou ASP).

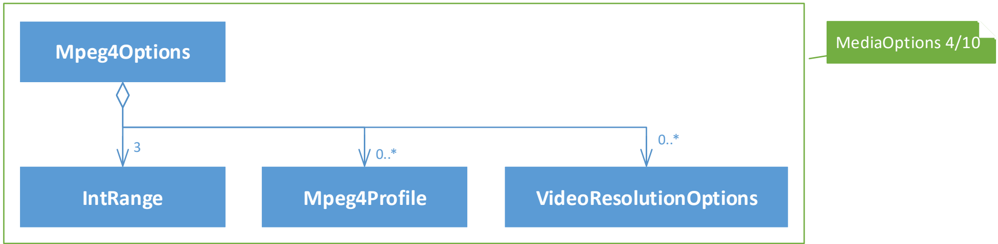


Figura V.0.34 - Diagrama de classes de dados para MPEG4 *options* do serviço Media

O diagrama de classes de dados das *media options* parte 5 de 10 para as H264 *options* pode ser consultado na Figura V.0.35. Neste diagrama existem parâmetros para a lista de resoluções (altura e largura), *frame rate* (máximo e mínimo), *encoding interval* (máximo e mínimo), GOV *length range* (máximo e mínimo) e lista dos perfis de H264 (podem ser Baseline, Main, Extended ou High).

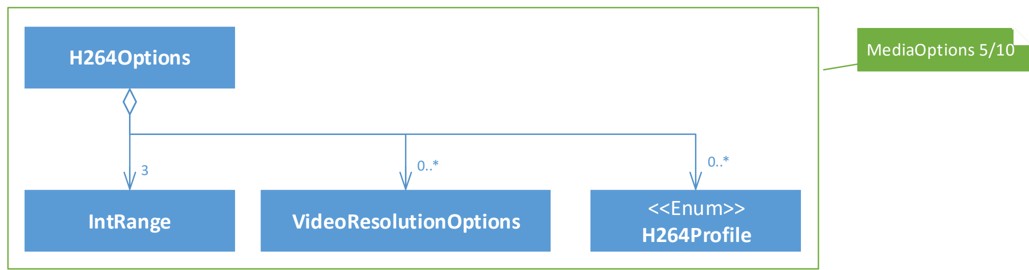


Figura V.0.35 - Diagrama de classes de dados para H264 *options* do serviço Media

O diagrama de classes de dados das *media options* parte 6 de 10 para as *video source configuration options* pode ser consultado na Figura V.0.36. Nas classes do diagrama existem parâmetros para o intervalo (máximo e mínimo) dos limites de um retângulo (x, y, altura, largura). Existe ainda uma lista para os *tokens* dos *video source* disponíveis.

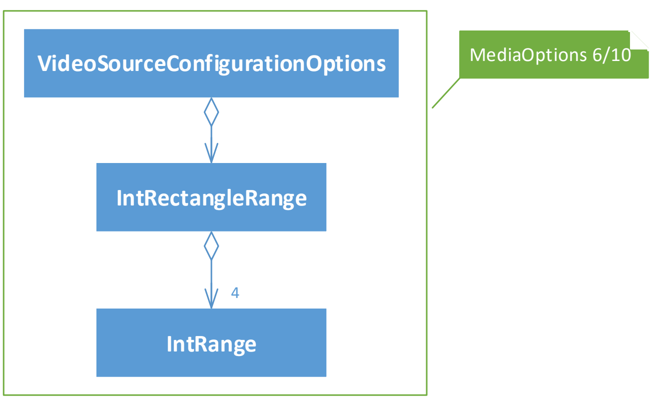


Figura V.0.36 - Diagrama de classes de dados para *video source configuration options* do serviço Media

O diagrama de classes de dados das *media options* parte 7 de 10 para as *audio encoder configuration options* pode ser consultado na Figura V.0.37. As classes do diagrama têm parâmetros

para lista de *audio encoder options* que contêm o *encoding* (G711/G726/AAC), conjunto de *bit rate e* conjunto de *sample rate*.

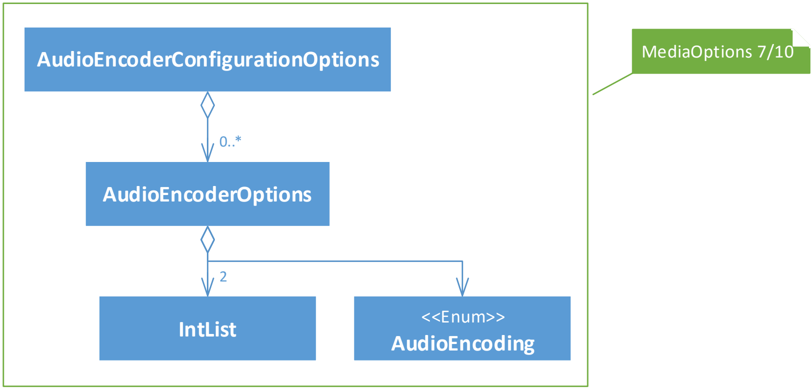


Figura V.0.37 - Diagrama de classes de dados para *audio encoder configuration options* do serviço Media

O diagrama de classes de dados das *media options* parte 8 de 10 para as PTZ *configuration options* pode ser consultado na Figura V.0.38. Os parâmetros deste diagrama consistem numa lista de configurações com parâmetros para o *token*, PTZ *spaces options*, PTZ *timeout* (mínimo e máximo) e PTZ *control direction options*.

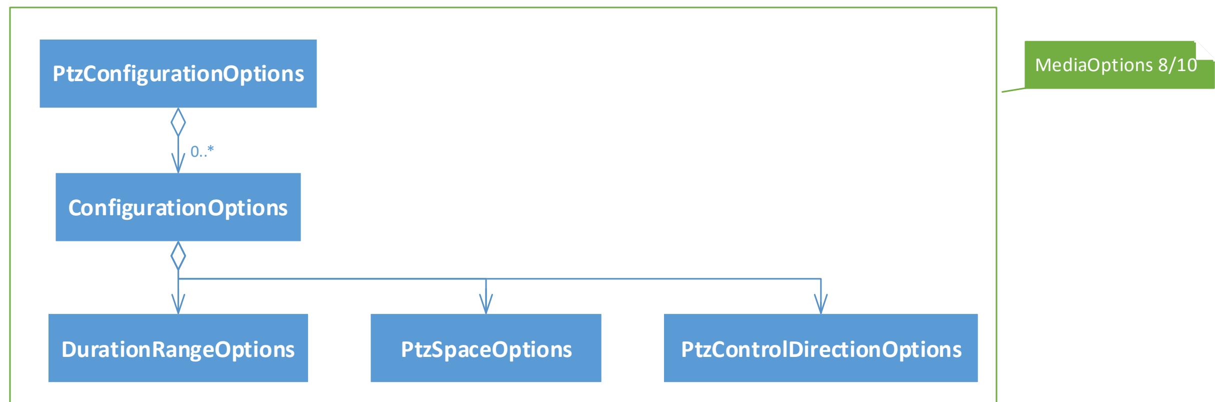


Figura V.0.38 - Diagrama de classes de dados para PTZ *configuration options* do serviço Media

O diagrama de classes de dados das *media options* parte 9 de 10 para as PTZ *space options* pode ser consultado na Figura V.0.39. Os parâmetros deste diagrama têm listas para os movimentos *absolute pan tilt position space, absolute zoom position space, relative pan tilt translation space, relative zoom translation space, continuous pan tilt velocity space, continuous zoom velocity space, pan tilt speed space, zoom speed space.* Os espaços do tipo 1D têm URI, x mínimo e x máximo enquanto o 2D é a adição do 1D com y mínimo e y máximo.

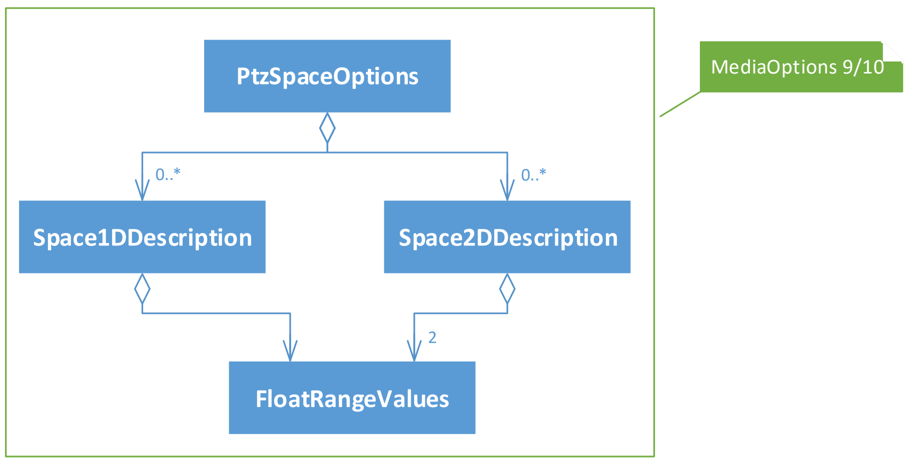


Figura V.0.39 - Diagrama de classes de dados para PTZ *space options* do serviço Media

O diagrama de classes de dados das *media options* parte 10 de 10 para as PTZ *control direction options* pode ser consultado na Figura V.0.40. Este diagrama tem parâmetros para as PTZ *control direction options* com a lista de EFlip *options modes* e a lista de *reverse options modes.*

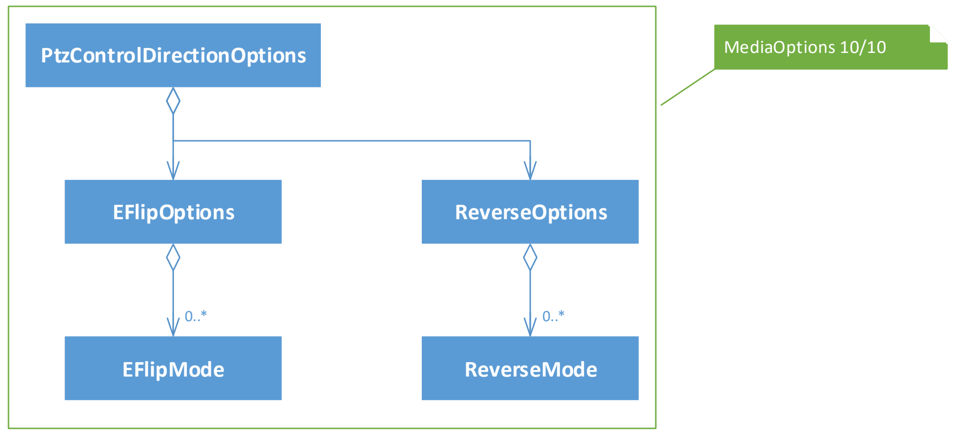


Figura V.0.40 - Diagrama de classes de dados para PTZ *control direction options* do serviço Media

O diagrama de classes de dados para o *media profile* foi dividido em 11 diagramas devido à sua complexidade, facilitando assim, a sua visualização.

O diagrama de classes de dados 1 de 11 para o *media profile* pode ser consultado na Figura V.0.41. Existem parâmetros para cada uma das classes do diagrama que são opcionais e respeitam as regras das opções de *media.* Existe ainda parâmetros para o *token* e se é *fixed.*

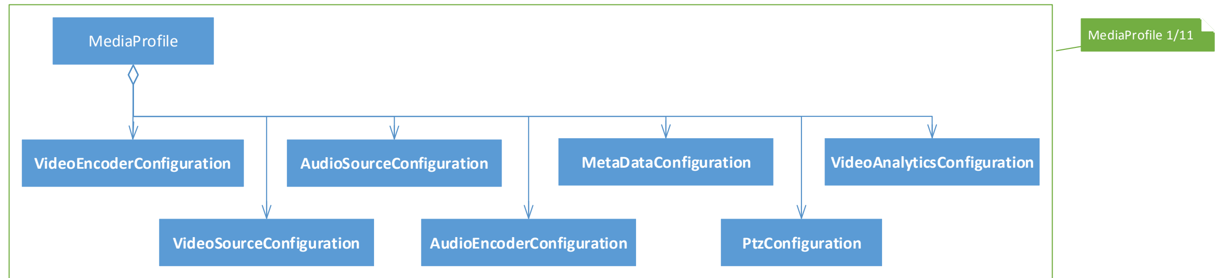


Figura V.0.41 - Diagrama de classes de dados para *media profile* do serviço Media

O diagrama de classes de dados 2 de 11 para a *video source configuration* pode ser consultado na Figura V.0.42. Os parâmetros das classes do diagrama consistem no *source token* e *bounds* (x, y, altura, largura).

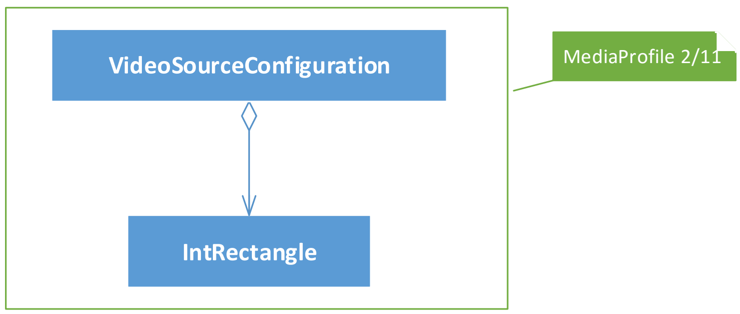


Figura V.0.42 - Diagrama de classes de dados para *video source configuration* do serviço Media

O diagrama de classes de dados 3 de 11 para a *video encoder configuration* pode ser consultado na Figura V.0.43. As classes do diagrama têm parâmetros para o *encoding*, *resolution*, *quality*, *rate control*, MPEG4/H264 *configuration information, multicast* e *session timeout*.

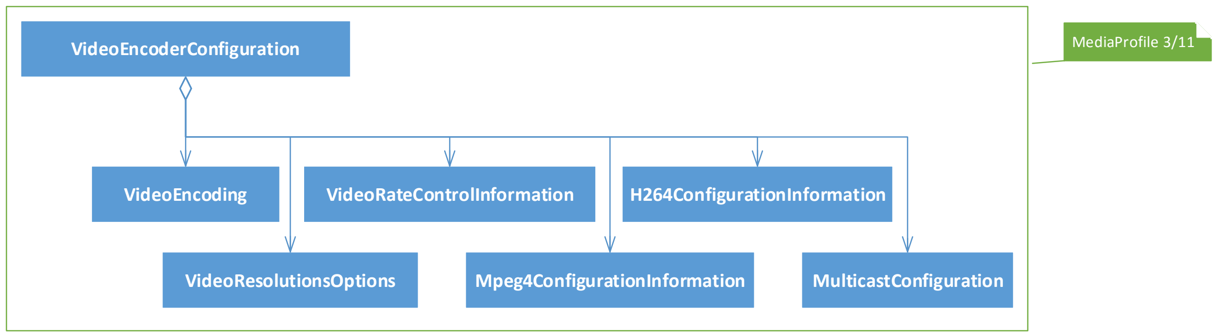


Figura V.0.43 - Diagrama de classes de dados para *video enconder configuration* do serviço Media

O diagrama de classes de dados 4 de 11 para as informações de MPEG4 e H264 *configuration* pode ser consultado na Figura V.0.44. As classes do diagrama têm parâmetros para o GOV *lenth,* MPEG4 *profile* (SP/ASP) e H264 *profile* (Baseline/Main/Extended/High).

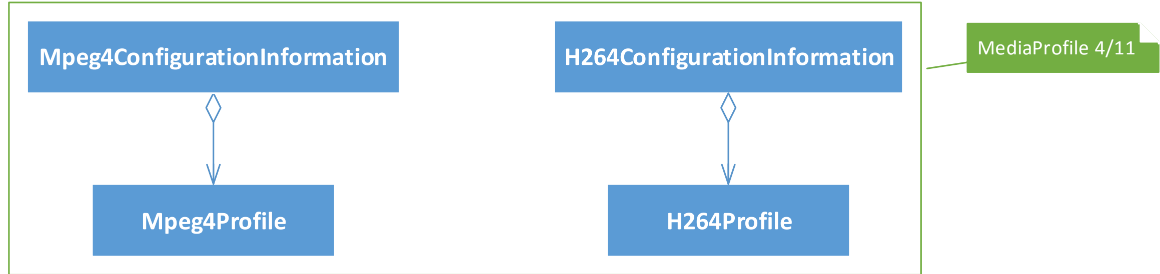


Figura V.0.44 - Diagrama de classes de dados para MPEG4/H264 *configuration information* do serviço Media

O diagrama de classes de dados 5 de 11 para a *multicast configuration* pode ser consultado na Figura V.0.45. As classes do diagrama têm parâmetros para o endereço IP e tipo de IP, *port*, TTL e *auto start*.

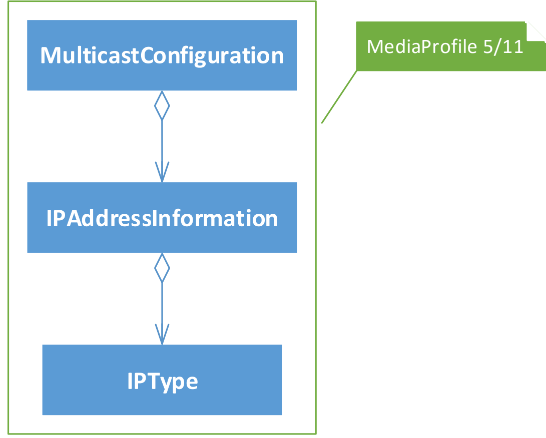


Figura V.0.45 - Diagrama de classes de dados para *multicast configuration* do serviço Media

O diagrama de classes de dados 6 de 11 para a *audio encoder configuration* pode ser consultado na Figura V.0.46. As classes do diagrama têm parâmetros para o *encoding, bit rate, sample rate, multicast* e *session timeout.*

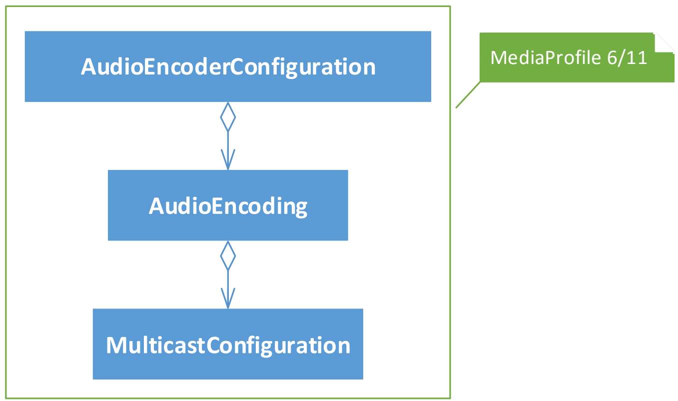


Figura V.0.46 - Diagrama de classes de dados para *audio enconder configuration* do serviço Media

O diagrama de classes de dados 7 de 11 para a *metadata configuration* pode ser consultado na Figura V.0.47. As classes do diagrama têm parâmetros para o PTZ *status* (estado e posição), *events* (lista de filtros e políticas de subscrições), *analytics, multicast, session timeout* e *analytics engine configuration* (tem diversos parâmetros para o serviço Analytics).

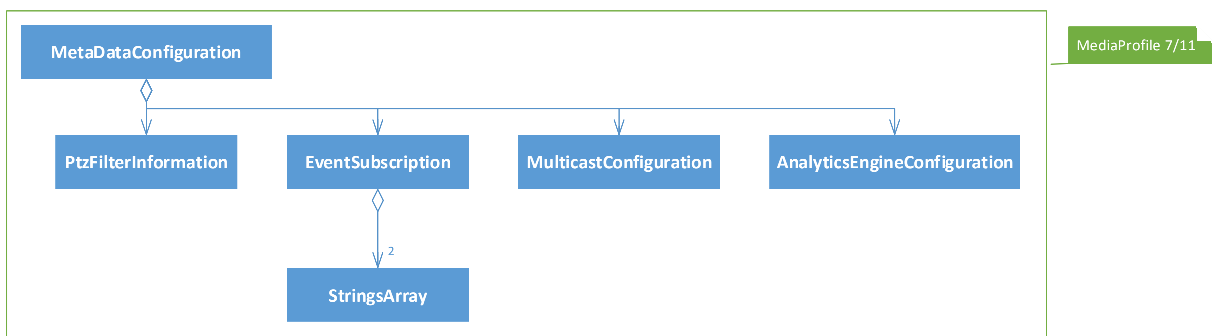


Figura V.0.47 - Diagrama de classes de dados para *metadata configuration* do serviço Media

O diagrama de classes de dados 8 de 11 para a *analytics engine configuration* pode ser consultado na Figura V.0.48. As classes do diagrama têm parâmetros para atributo, lista de *analytics module, lista de analytics engine configuration extension.* Existem ainda diversos parâmetros para lista de extensões, lista de itens simples e elementares.

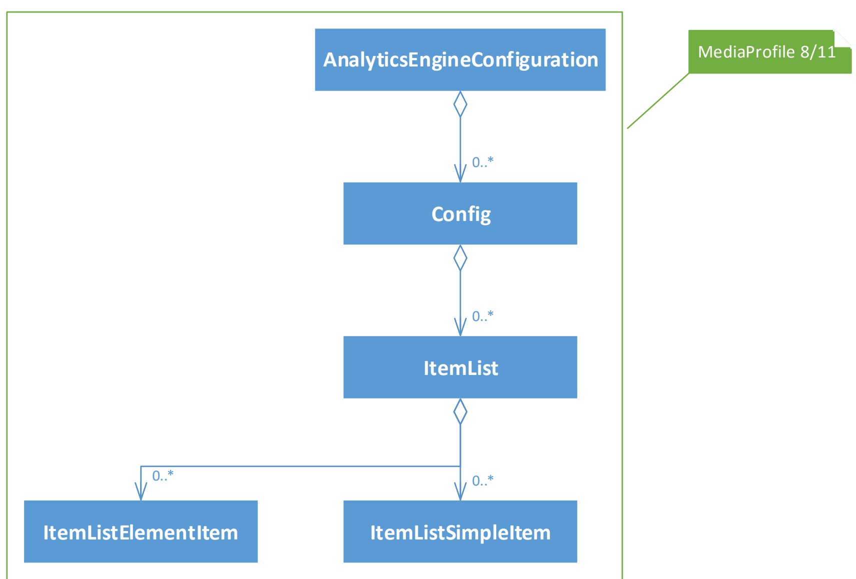


Figura V.0.48 - Diagrama de classes de dados para *analytics engine configuration* do serviço Media

O diagrama de classes de dados 9 de 11 para a PTZ *configuration* pode ser consultado na Figura V.0.49. As classes do diagrama têm parâmetros semelhantes aos das opções de *media* para os valores por defeito. Existe ainda parâmetros para o *token* e PTZ *timeout.*

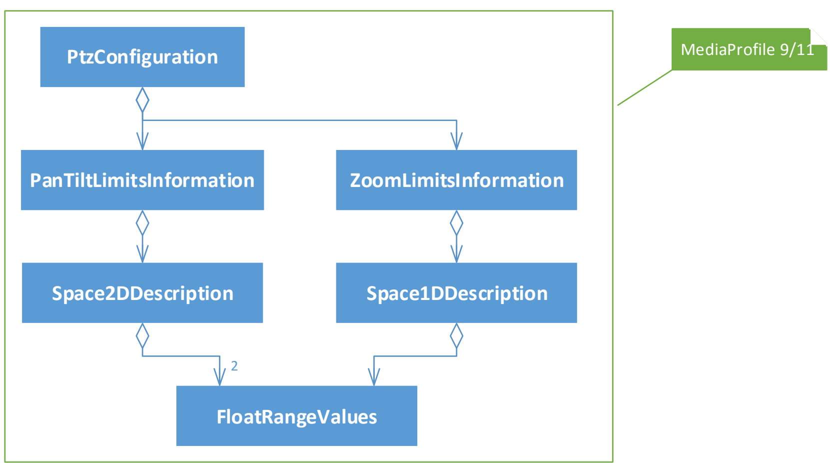


Figura V.0.49 - Diagrama de classes de dados para PTZ *configuration* do serviço Media

O diagrama de classes de dados 10 de 11 para a *video analytics configuration* pode ser consultado na Figura V.0.50. As classes do diagrama têm parâmetros para a *analytics engine configuration* e *rule engine configuration*.

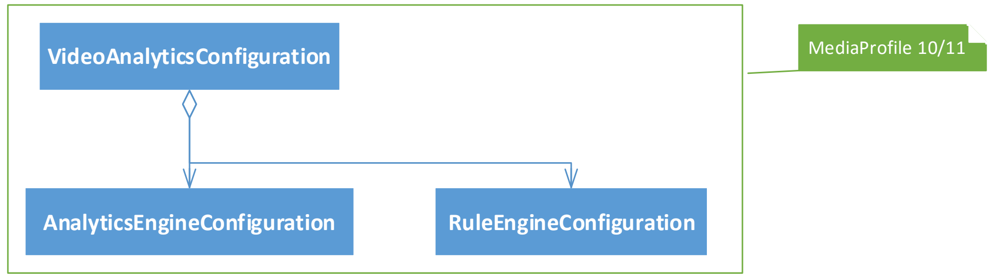


Figura V.0.50 - Diagrama de classes de dados para *video analytics configuration* do serviço Media

O diagrama de classes de dados 11 de 11 para a *rule engine configuration* pode ser consultado na Figura V.0.51. As classes do diagrama têm parâmetros a *rule*, *rule engine configuration extension* e *attribute*.

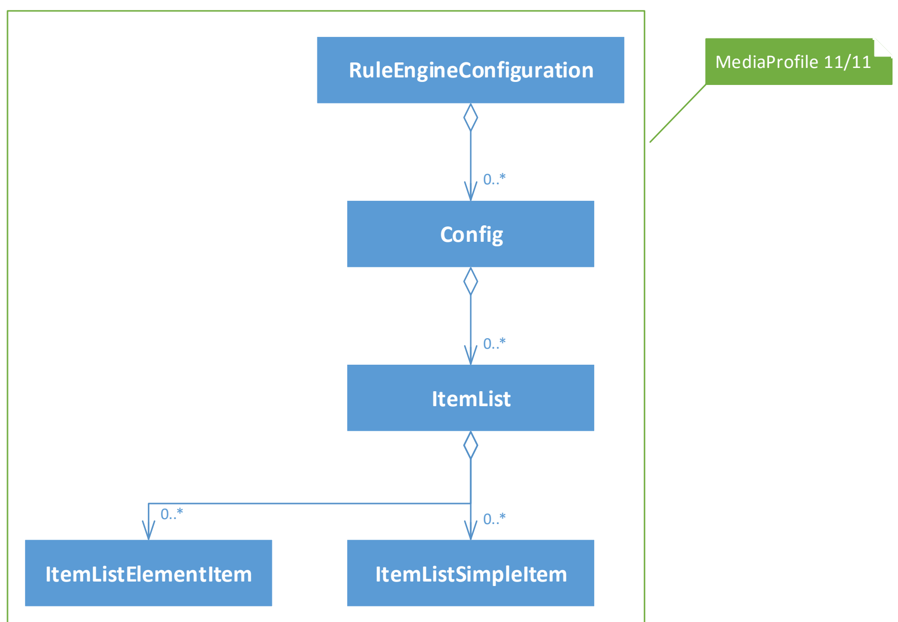


Figura V.0.51 - Diagrama de classes de dados para rule *engine configuration* do serviço Media

PTZ

O diagrama de classes de dados das *capabilities* para PTZ pode ser consultado na Figura V.0.52. As *capabilities* consistem nos parâmetros opcionais *eFlip* e *reverse*.

Figura V.0.52 - Diagrama de classes de dados para *capabilities* do serviço PTZ

O diagrama de classes de dados para os movimentos contínuos pode ser consultado na Figura V.0.53. Os parâmetros das classes do diagrama são o *timeout, pan tilt speed* (x, y, *space*) e *zoom speed* (x, *space*).

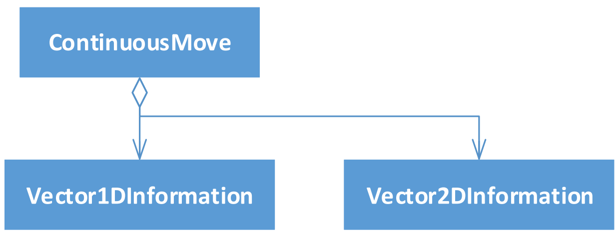


Figura V.0.53 - Diagrama de classes de dados para *continuous* move do serviço PTZ

O diagrama de classes de dados para os movimentos absolutos pode ser consultado na Figura V.0.54. As classes do diagrama têm parâmetros para *pan tilt position* (x, y, *space*), *zoom position* (x, *space*), *pan tilt speed* (x, y, *space*) *e zoom speed* (x, *space*).

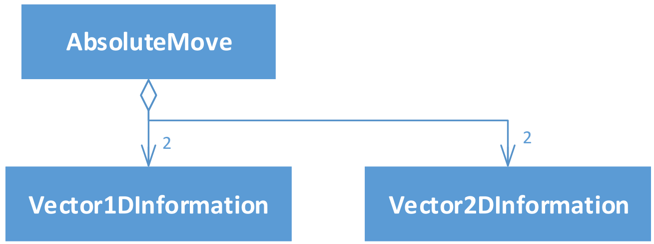


Figura V.0.54 - Diagrama de classes de dados para *absolute move* do serviço PTZ

O diagrama de classes de dados para os movimentos relativos pode ser consultado na Figura V.0.55. As classes do diagrama têm parâmetros para *pan tilt shift* (x, y, *space*), *zoom shift* (x, *space*), *pan tilt speed* (x, y, *space*) *e zoom speed* (x, *space*).

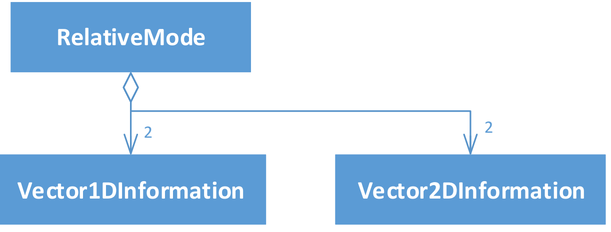


Figura V.0.55 - Diagrama de classes de dados para *relative mode* do serviço PTZ

O diagrama de classes de dados para as informações de estado PTZ pode ser consultado na Figura V.0.56. As classes do diagrama têm parâmetros para a *pan tilt/zoom position* (x, y, *space* e x, *space*), *pan tilt/zoom move status* (IDLE/MOVING e UNKNOW) e *error*.

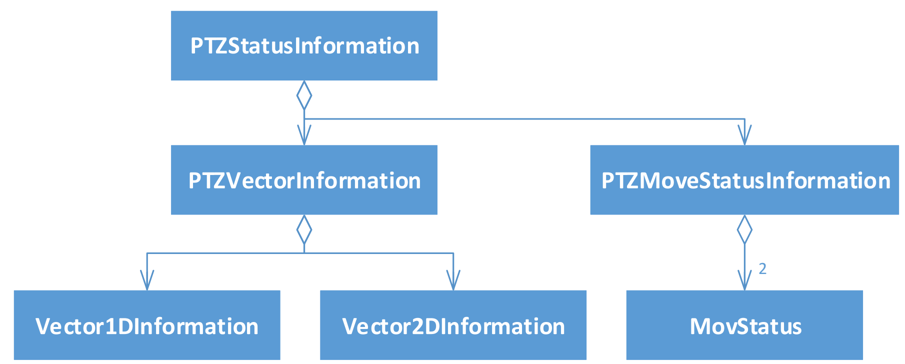


Figura V.0.56 - Diagrama de classes de dados para PTZ *status information* do serviço PTZ

ANEXO VI – CLASSES DE SERVIÇO

O diagrama de classes de serviço para o Device Managment é exibido na Figura VI.0.1 onde podem ser consultados os construtores e métodos implementados.

Figura VI.0.1 - Diagrama de classes de serviço do Device Mamagment

O diagrama de classes de serviço para o Discovery é exibido na Figura VI.0.2 onde podem ser consultados os construtores. Como o serviço não se encontra implementado os métodos estão por definir.

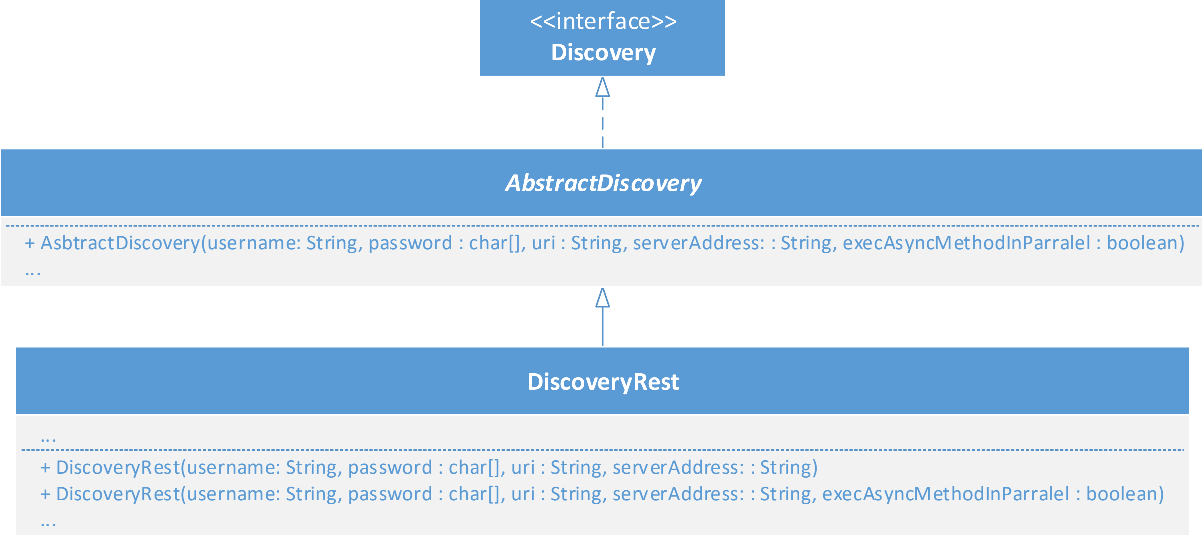


Figura VI.0.2 - Diagrama de classes de serviço do Discovery

O diagrama de classes de serviço para o Events é exibido na Figura VI.0.3 onde podem ser consultados os construtores. Como o serviço não se encontra implementado os métodos estão por definir. O trabalho futuro requer a integração do serviço Events no servidor e do uso de SSE (server sent evnts) na biblioteca.

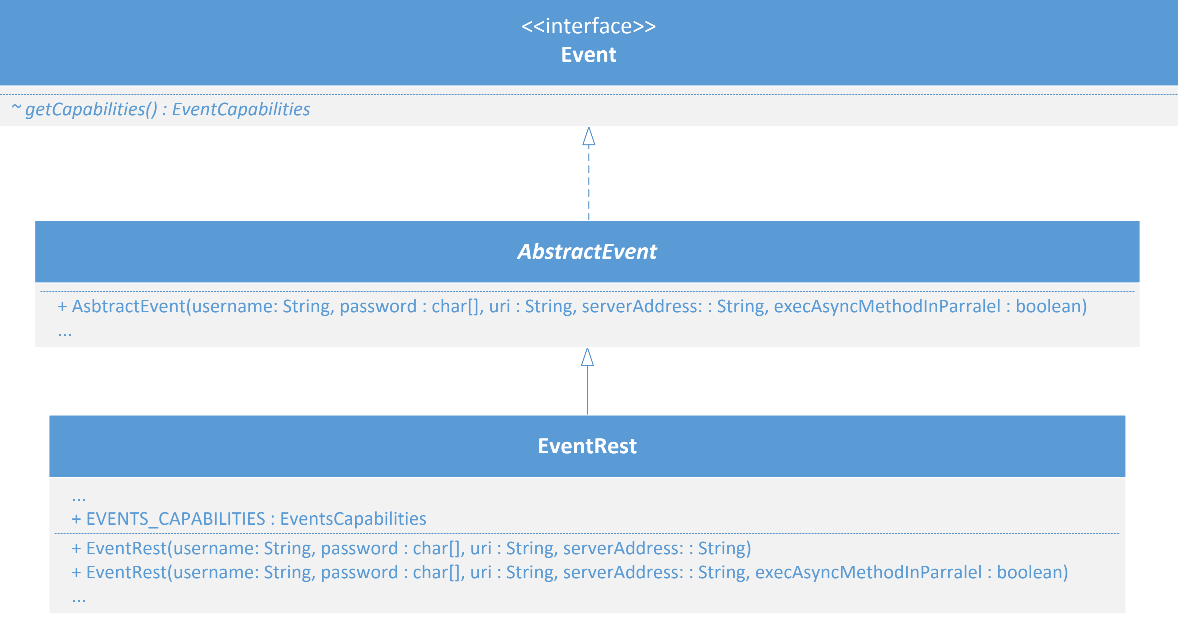


Figura VI.0.3 - Diagrama de classes de serviço do Events

O diagrama de classes de serviço para o Imaging é exibido na Figura VI.0.4 onde podem ser consultados os construtores e métodos implementados.

Figura VI.0.4 - Diagrama de classes de serviço do Imaging

O diagrama de classes de serviço para o Media é exibido na Figura VI.0.5 onde podem ser consultados os construtores e métodos implementados.

Figura VI.0.5 - Diagrama de classes de serviço do Media

O diagrama de classes de serviço para o PTZ é exibido na Figura VI.0.6 onde podem ser consultados os construtores e métodos implementados.

Figura VI.0.6 - Diagrama de classes de serviço do PTZ

ANEXO VII – ANÁLISE TRAFEGO

Para analisar eficazmente o trafego das aplicações que comunicam SOAP para ONVIF é necessário entender como estão estruturados os protocolos de comunicação. O Wireshark tem ferramentas que ajudam a analisar tráfego capturado e a interpretar o mesmo.

A Figura VII.0.1 representa as várias camadas utilizadas no pedido HTTP Request para estruturar dados. Tanto o SOAP envelope como o HTTP Request, tem Header e Body, sendo que o SOAP envelope está embebido no HTTP Body.

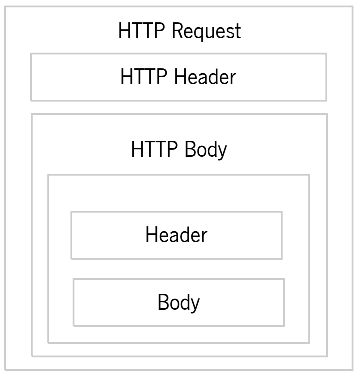


Figura VII.0.1 - Estrutura dos pedidos HTTP do tipo POST

A Figura VII.0.2 é um exemplo do formato de dados do HTTP Request para o pedido GetSystemDateAndTime. O cabeçalho tem informações sobre o tipo de pedido, URI utilizado, versão do protocolo HTTP, agente, *encoding,* tamanho do conteúdo e endereço de destino da mensagem. No conteúdo de corpo estão as informações relativas ao SOAP envelope.

```
POST /onvif/device_service HTTP/1.1
User-Agent: ksoap2-android/2.6.0+
Content-Type: application/soap+xml;charset=utf-8
Accept-Encoding: gzip
Connection: close
Content-Length: 471
Host: 193.136.12.31
```

```
<v:Envelope xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns:d="http://www.w3.org/2001/XMLSchema" xmlns:c="http://
www.w3.org/2003/05/soap-encoding" xmlns:v="http://www.w3.org/
2003/05/soap-envelope"><v:Header><Action mustUnderstand="1"
xmlns="http://www.w3.org/2005/08/addressing">http://
www.onvif.org/ver10/device/wsdl/GetSystemDateAndTime</Action></
v:Header><v:Body><GetSystemDateAndTime xmlns="http://
www.onvif.org/ver10/device/wsdl" /></v:Body></v:Envelope>
```


Figura VII.0.2 - Formato de dados do HTTP Request

O HTTP Response tem cabeçalho e corpo tal como o HTTP Request. O cabeçalho tem informações relevantes como o código da resposta que é útil para verificar a existência de problemas

na comunicação. O SOAP envelope está contido no conteúdo de corpo. Um exemplo do formato do HTTP Response pode ser consultado na Figura VII.0.3.

```
HTTP/1.1 200 OK
Date: Tue, 09 Feb 2016 14:52:40 GMT
Server: App-webs/
Connection: close
Content-Length: 2675
Content-Type: application/soap+xml; charset=utf-8
```
Header

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope" xmlns:soapenc="http://www.w3.org/2003/05/soap-
encoding" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tt="http://
www.onvif.org/ver10/schema" xmlns:tds="http://www.onvif.org/
ver10/device/wsdl" xmlns:trt="http://www.onvif.org/ver10/media/
wsdl" xmlns:timg="http://www.onvif.org/ver20/imaging/wsdl"
xmlns:tev="http://www.onvif.org/ver10/events/wsdl"
xmlns:tptz="http://www.onvif.org/ver20/ptz/wsdl"
xmlns:tan="http://www.onvif.org/ver20/analytics/wsdl"
xmlns:tst="http://www.onvif.org/ver10/storage/wsdl"
...
```
Body

Figura VII.0.3 - Formato de dados do HTTP Response

A Figura VII.0.4 mostra o conteúdo pertencente ao SOAP envelope body do pedido GetSystemDateAndTime. A partir deste campo é possível observar as informações relativas ao pedido.

```
schema"><env:Body><tds:GetSystemDateAndTimeResponse><tds:SystemD
ateAndTime><tt:DateTimeType>NTP</tt:DateTimeType>
<tt:DaylightSavings>true</tt:DaylightSavings>
<tt:TimeZone><tt:TZ>CST
+0:00:00DST01:00:00,M4.1.0/02:00:00,M10.5.0/02:00:00</tt:TZ>
</tt:TimeZone>
<tt:UTCDateTime><tt:Time><tt:Hour>14</tt:Hour>
<tt:Minute>52</tt:Minute>
<tt:Second>40</tt:Second>
</tt:Time>
<tt:Date><tt:Year>2016</tt:Year>
<tt:Month>2</tt:Month>
<tt:Day>9</tt:Day>
</tt:Date>
</tt:UTCDateTime>
<tt:LocalDateTime><tt:Time><tt:Hour>14</tt:Hour>
<tt:Minute>52</tt:Minute>
<tt:Second>40</tt:Second>
</tt:Time>
<tt:Date><tt:Year>2016</tt:Year>
<tt:Month>2</tt:Month>
<tt:Day>9</tt:Day>
</tt:Date>
</tt:LocalDateTime>
</tds:SystemDateAndTime>
</tds:GetSystemDateAndTimeResponse>
```

Figura VII.0.4 - Conteúdo do envelope do pedido GetSystemDateAndTime

ANEXO VIII – PROCESSO INSTALAÇÃO SERVIDOR WEB REST

Instalar o apache

No Ubuntu a instalação de pacotes pode ser feita com recurso ao assistente de instalação de pacotes ou através da linha de comandos. O comando seguinte permite instalar um ou mais pacotes:

sudo apt-get instal [*packet1 packet2 ...*] [74].

A instalação mais recente do servidor *web* Apache pode ser efetuada executando o comando:

sudo apt-get install apache2

Opcionalmente é possível instalar a documentação e as utilidades do Apache através dos pacotes *apache2-doc* e *apache2-utils*.

Gerar um certificado SSL

Para ativar o módulo ssl (*Secure Sockets Layer*) do apache2 tem-se o comando:

sudo a2enmod ssl

Para desativar um módulo usa-se o comando:

sudo a2dismod [nome do módulo]

Foi gerado e instalado um *self-signed certificate*, ou seja, um certificado assinado pelo próprio servidor, em vez de uma *certificate authority* que é uma entidade confiável que pode emitir certificados. Para gerar o certificado referido tem-se o comando:

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

Na criação do certificado "restonvif.crt" é requerido o nome/endereço do site.

Configurar o sítio web

O próximo passo consiste em criar e configurar o sítio *web* que irá alojar o SW REST instalando o certificado SSL criado. Inicialmente devem ser realizados os seguintes passos:

- Criar um novo ficheiro de configuração na pasta *sites-available*. Para facilitar pode-se criar uma copia do ficheiro de configurações do sítio *web* por defeito (default-ssl.conf), e adaptá-las ao novo sitio/serviço que se pretende configurar.
- Nomear o ficheiro para "ssl-restonvif.pt";
- Ativar o *site* executando o comando: *sudo a2ensite ssl-restonvif.pt*. Para desativar um *site* usa-se o comando *sudo a2dissite* [nome do site].
- Por último, fazer o *reload* do apache2 como o comando "*sudo service apache2 reload*" de modo a efetuar as novas configurações.

Nesta fase para o normal funcionamento, o ficheiro de configurações do sítio *web* deve conter as configurações:

```
<VirtualHost _default_:443>
        ServerName restonvif.pt
        AllowEncodedSlashes NoDecode
        SSLProtocol all
        SSLEngine on
        SSLCertificateFile /etc/ssl/private/restonvif.crt
...
```

A razão da utilização para cada uma das diretivas é a seguinte:

- **ServerName:** especifica o nome/endereço para o sítio *web* respetivo ao *hostname* configurado na criação do certificado.
- **AllowEncodedSlashes:** permite aos URLs conter no caminho separadores codificados (e.g., "%2F" para o carater "/") para serem usados na informação do caminho. Com o valor NoDecode, os URLs são aceites, mas os carateres codificados não são descodificados e deixados na forma original. Se apenas os carateres codificados forem necessários na informação do caminho é recomendado o uso do NoDecode como medida de segurança. Permitir que o carater "/" seja descodificado pode permitir potenciais caminhos inseguros. Este parâmetro é útil porque o endereço de serviço da câmara é codificado.
- **SSLProtocol:** configura os diversos tipos de protocolos SSL. Os clientes só podem estabelecer ligação com os protocolos providenciados. A opção *all* permite configurar todos os protocolos SSL fornecidos pelo *mod_ssl*, respetivamente o SSLv2, SSLv3, TLSv1, TLSv1.1 e TLSv1.2.
- **SSLEngine:** por defeito todos os *virtual hosts* têm o SSL/TLS *protocol engine* desativado para os servidores principais e para todos os virtual *hosts* configurados. Esta diretiva permite ativar o SSL/TLS *protocol engine*.
- **SSLCertificateFile:** esta diretiva aponta para o certificado no formato PEM, neste caso, o certificado criado.

Configuração de logs

Há dois ficheiros de *log* essenciais do Apache que têm o nome por defeito de "error.log" e "access.log" e estão configurados para serem guardados na diretoria de *logs* do apache em "/var/log/apache2/". O "error.log" é um ficheiro importante uma vez que recebe e guarda informação

relativa a erros ou eventos, tais como tentativa de avesso a um ficheiro inexistente, quando processa pedidos. O *log* de acessos regista todos os pedidos ao servidor e contém informações acerca do cliente.

Nas configurações do sítio *web* é possível registar os erros e acessos ocorridos nos ficheiros do apache através das linhas seguintes:

```
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
```

As diretivas usadas são:

- **ErrorLog**: para configurar o registo de erros.
- **CustomLog**: para configurar o registo de acessos.

Instalar o FastCGI

Para executar um ficheiro fcgi é necessário instalar o módulo fastcgi para executar o serviço REST ONVIF que consiste no ficheiro binário umoc_rest.fcgi. Após a instalação com o comando *apt-get install*, o módulo fastcgi fica disponível na lista de módulos ativos. Para ser ativado deve ser usado o comando para ativar o módulo:

sudo a2enmod fastcgi

O ficheiro de configuração do módulo fastcgi fica disponível na pasta mods-enabled do Apache. Inicialmente o ficheiro contém as seguintes configurações:

```
<IfModule mod_fastcgi.c>
        AddHandler fastcgi-script .fcgi
        FastCgiIpcDir /var/lib/apache2/fastcgi
</IfModule>
```

A razão da utilização de cada uma das diretivas é a seguinte:

- **AddHandler:** ficheiros com uma determinada extensão são tratados pelo nome do *handler*. Na configuração acima qualquer programa com a extensão fcgi é tratado como um programa FastCGI.
- **FastCgiIpcDir:** específica a diretoria para armazenar os *sockets* Unix usados para a comunicação entre aplicações e o servidor *web*.

Adicionou-se a linha de configuração seguinte que corresponde as "*Dynamic FastCGI applications*".

```
FastCgiConfig -maxProcesses 50 -maxClassProcesses 20 -startDelay 1 -killInterval 30 -idle-timeout 90
```

A diretiva FastCgiConfig permite definir os parâmetros padrão para as aplicações FastCGI. O significado de cada um dos parâmetros é o seguinte:

- **-maxProcesses**: Corresponde ao número total máximo de instâncias de aplicações FastCGI dinâmicas permitidas para execução a qualquer momento. Deve ser maior ou igual que o parâmetro –maxClassProcesses.
- **-maxClassProcesses**: Corresponde ao número total máximo de instâncias de aplicações FastCGI dinâmicas permitidas para execução de qualquer aplicação FastCGI. Deve ser menor ou igual que o parâmetro –maxProcesses.
- **-startDelay:** O número de segundos que o servidor *web* espera enquanto tenta se conectar a uma aplicação FastCgi dinâmica.
- **-killInterval:** Define o intervalo em segundos com que uma instância de uma aplicação dinâmica é sujeita à política de terminar processos aplicada pelo gestor de processos.
- **-idle-timeout:** Configura o número de segundos de inatividade permitidos da aplicação FastCgi antes do pedido ser abortado e o evento é registado. O temporizador de inatividade aplica-se enquanto uma conexão estiver pendente com a aplicação FastCGI. Se durante o intervalo de tempo de inatividade a aplicação não responder, o pedido é abortado. Se a comunicação esta completa com a aplicação, mas incompleta com o cliente a resposta é armazenada, o *timeout* não se aplica.

Configurações adicionadas ao sítio web:

```
Action fastcgi-script /home/servidor/umoc_rest.fcgi
ScriptAlias /onvif /home/servidor/umoc_rest.fcgi
<Directory "/home/servidor">
        AllowOverride All
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Require all granted
        Script GET umoc_rest.fcgi
        Script POST umoc_rest.fcgi
        Script PUT umoc_rest.fcgi
        Script DELETE umoc_rest.fcgi
</Directory>
```

A última parte passa por adicionar diretivas ao ficheiro de configuração *ssl-restonvif.pt*. para indicar a diretoria do binário FastCGI [25], [26] e mapear os caminhos das URI do serviço que serão tratadas pelo FastCGI e controlar o acesso. Neste contexto as diretivas têm o significado seguinte:

- **Action:** acrescenta uma ação, que ativara o ficheiro umoc_rest.fcgi quando o fastcgi-script for acionado por um pedido. É necessário instalar o pacote mod_actions e certificar a ativação do módulo *action*.
- **ScriptAlias:** os pedidos feitos a https://restonvif.pt/onvif/ são tratados pelo ficheiro umoc_rest.fcgi. deve-se certificar a ativação do módulo *alias*.
- **AllowOverride:** quando esta diretiva é definida com o valor *all,* qualquer diretiva que tenha *.htaccess Context* é permitida nos ficheiros *.htaccess*. *Context* é usado pelo Apache para indicar em qual ficheiro de configurações do servidor a diretiva é legal.
- **Options:** permite controlar quais as funcionalidades do servidor que são disponibilizadas. A opção ExecCGI permite a execução de *scripts* cgi e o mod_fastcgi requer a ativação desta opção.
- **Require all granted:** permite aceitar todos os pedidos.
- **Script:** serve para o servidor redirecionar os 4 tipos de pedidos suportados pelo serviço *web* REST para a aplicação FastCGI (umoc_rest.fcgi). É necessário a instalação do pacote mod_rewrite e certificar a ativação do módulo *rewrite*.

Para que o ficheiro umoc_rest.fcgi seja executado é necessário ceder permissões de execução. São atribuídas as permissões com o comando: *sudo chmod +x /home/servidor/umoc_rest.fcgi*

Dependências

O serviço web FastCGI tem dependências de bibliotecas de linkagem dinâmica, nomeadamente “libjson”, “libfcgi” e “libssl”. Geralmente estas não se encontram instaladas no sistema, o que faz com que o arranque do serviço falhe. Numa primeira fase, os pedidos de teste do servidor falhavam, o que requereu a análise dos ficheiros de *logs.* Verificou-se que os acessos estavam a ser feitos, mas que o módulo fastcgi falhava em iniciar e que existiam dependências. A descrição das dependências de bibliotecas é encaminhada para o ficheiro de erros do apache. Verificou-se que, para o normal funcionamento do servidor, são necessários os pacotes “libjson”, “libfcgi” e “libssl” que podem ser instalados com recurso ao comando de instalar pacotes ou através de um gestor de instalações de pacotes do sistema operativo. Para instalar através de comandos recomenda-se

executar o comando "*sudo apt-get update*" para atualizar a lista de pacotes do sistema e o comando "*sudo apt-get upgrade*" para efetuar uma atualização antes da instalação das dependências.

ANEXO IX - RESULTADOS DA ANÁLISE DO TIPO DE PROGRESSO

A Tabela IX.0.1 contém os valores dos tempos analisados para o pedido GetMediaProfile, assim como, o cálculo do tempo total e de rede.

Tabela IX.0.1 - Tempos em milissegundos para o pedido GetMediaProfile

Tempo (ms) / Nº Pedido	Serialização	Conexão	Receção	Desserialização	Total	Rede
1	0	0,211667	716,999427	28,402865	745,613959	717,211094
2	0	0,306875	746,967395	48,956667	796,230937	747,27427
3	0	0,21099	698,493802	50,385469	749,090261	698,704792
4	0	0,210521	565,974896	37,872083	604,0575	566,185417
5	0	0,380469	779,434583	63,993542	843,808594	779,815052
6	0	0,368385	734,915624	57,470001	792,75401	735,284009
7	0	0,210937	493,230364	38,826511	532,267812	493,441301
8	0	0,301042	680,694999	49,116406	730,112447	680,996041
9	0	0,260156	689,301302	51,306094	740,867552	689,561458
10	0	0,275364	717,799374	59,482709	777,557447	718,074738
Média	0	0,2736406	682,3811766	48,5812347	731,236052	682,654817

Os resultados dos testes para o pedido SetSystemDateAndTime estão registados na Tabela IX.0.2 que contém os valores dos tempos, o cálculo do tempo total e de rede.

Tabela IX.0.2 - Tempos em milissegundos para o pedido SetSystemDateAndTime

Tempo (ms) / Nº Pedido	Serialização	Conexão	Envio	Receção	Desserialização	Total	Rede
1	0,079114	0,20349	48,000417	1325,43312	0,125782	1373,84193	1373,63703
2	0,080312	0,259896	49,227552	1081,88297	0,658593	1132,10932	1131,37042
3	0,060782	0,198125	30,046093	841,058959	0,188906	871,552865	871,303177
4	0,082344	0,236875	52,668072	1058,93031	0,187396	1112,105	1111,83526
5	0,082292	0,256198	47,890104	804,031094	0,119843	852,379531	852,177396
6	0,08125	0,252813	30,232812	1445,4663	0,177656	1476,21083	1475,95193
7	0,080468	0,225573	48,142969	1130,7187	0,175209	1179,34292	1179,08724
8	0,089063	0,224479	31,468489	757,170782	0,181927	789,13474	788,86375
9	0,08125	0,235521	47,975572	975,68151	0,189584	1024,16344	1023,8926
10	0,08177	0,223542	43,198959	1279,51115	0,158437	1323,17385	1322,93365
Média	0,0798645	0,2316512	42,8851039	1069,98849	0,2163333	1113,40144	1113,10524

ANEXO X – FUNCIONALIDADES E MENUS DA NOVA APLICAÇÃO

Menu inicial para gerir Câmaras

Ao iniciar a aplicação é apresentado o menu da Figura X.0.2. Neste menu é possível adicionar/editar/remover câmaras e alterar configurações gerais da aplicação. As Câmaras adicionadas são apresentadas em lista. Ao aplicar um *long tap* numa câmara é apresentado um AlertDialog com opções de editar e eliminar a câmara. Um *tap* numa câmara faz o avanço para o menu de gestão de serviços da câmara. Para adicionar uma câmara é utilizado o botão representado pelo sinal de mais que abre um AlertDialog para preencher os dados. Os dados de uma câmara são o endereço do servidor, nome da câmara, nome de utilizador, palavra-chave e endereço da câmara, sendo estes, editáveis. Caso o utilizador tente aceder a uma câmara *offline* é emitido um aviso num AlertDialog a pedir para escolher uma câmara *online*.

A Figura X.0.1 representa o Dialog de adição de câmaras.

As configurações gerais são acedidas através da Toolbar no botão com três pontos que são:

- **Update**, para verificar novamente o estado de conexão das câmaras. Os estados possíveis são *on* e *off* representados por dois ícones distintos.
- **About**, para consulta de informações sobre o desenvolvimento da aplicação. É feita uma breve explicação sobre o âmbito de desenvolvimento e a sua finalidade.
- **Definitions**, para configurar definições globais, como ativar o armazenamento local e definir o máximo de memória permitida para uso da aplicação. Estas opções são apresentadas num AlertDialog. Quando o armazenamento local está desativado a opção de definir o máximo de memória utilizada desaparece.
- **Delete Data**, para eliminar todos os dados das câmaras guardados pela aplicação. É pedida a confirmação ao utilizador para eliminar os dados

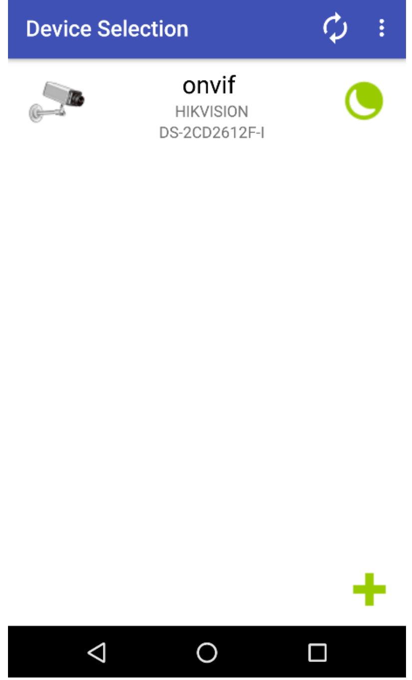


Figura X.0.2 - Menu inicial de gestão de câmaras

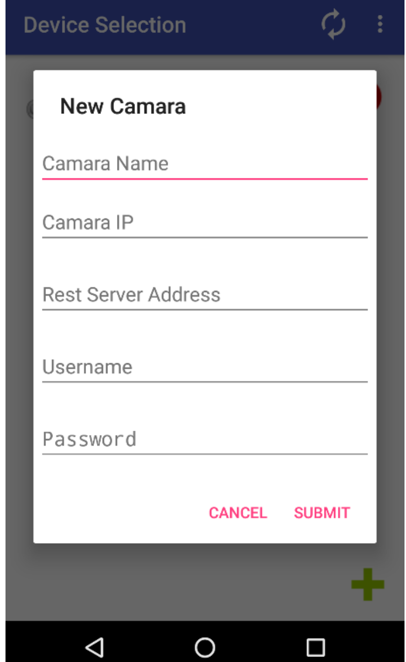


Figura X.0.1 - *Dialog* de adição de câmaras

O Dialog da opção Definitions da Toolbar é dinâmico de modo a habilitar a configuração da utilização de memória quando o armazenamento local está ativo como ilustrado na Figura X.0.3 e Figura X.0.4. As definições são guardadas em Shared Preferences [75],[76] porque são ideais para dados chave-valor que ficam guardados quando se fecha a aplicação.

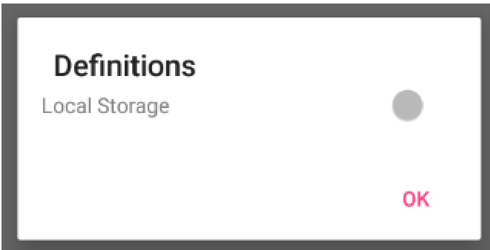


Figura X.0.3 - Dialog Definitions com o Local Storage desabilitado

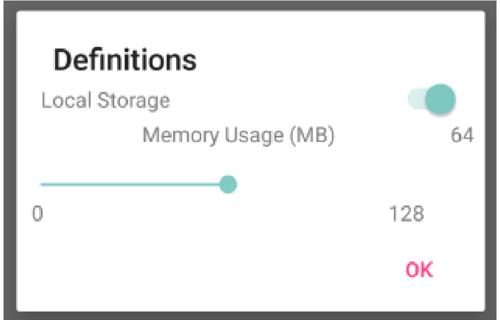


Figura X.0.4 - Dialog Definitions com o Local Storage habilitado

Menu Device Managment

O menu Device Managment está dividido em ecrãs System, Network e Users acedíveis através de tabuladores. Na Toolbar estão disponíveis opções para consultar as *capabilities* do *network*, *system*, *security* e *miscellaneous*.

O tabulador *system* apresenta informações para data, hora e scopes, os quais, estão divididos em dois cartões como ilustrado na Figura X.0.7. Estes dados são editáveis e têm ícones para efetuar ações na interface de utilizador como definido. Existe ainda uma opção para fazer *reboot* à câmara. O primeiro cartão diz respeito às operações de *reboot* e de data/hora e o segundo cartão faz a listagem de scopes com opções para adicionar, remover e editar. A data e a hora usam *widgets* do calendário e relógio para fazer as configurações.

O tabulador Network da Figura X.0.6. permite consultar e editar as informações de rede para o NTP, DNS e DNS dinâmico.

O tabulador Users representado na Figura X.0.5 permite consultar os utilizadores existentes e fazer a respetiva adição/edição/remoção. O utilizador "admin" não pode ser removido porque tem privilégios de administrador. Para criar/editar um utilizador é necessário introduzir o nome de utilizador, a palavra-chave e submeter o pedido.

A nível de alterações na UI foram eliminados textos redundantes e reajustadas as posições dos ícones. No caso do tabulador *system* acrescentou-se os cartões. Os ícones anteriores para edição e para eliminar não tinham significado direto pelo foram modificados para serem intuitivos e usuais.

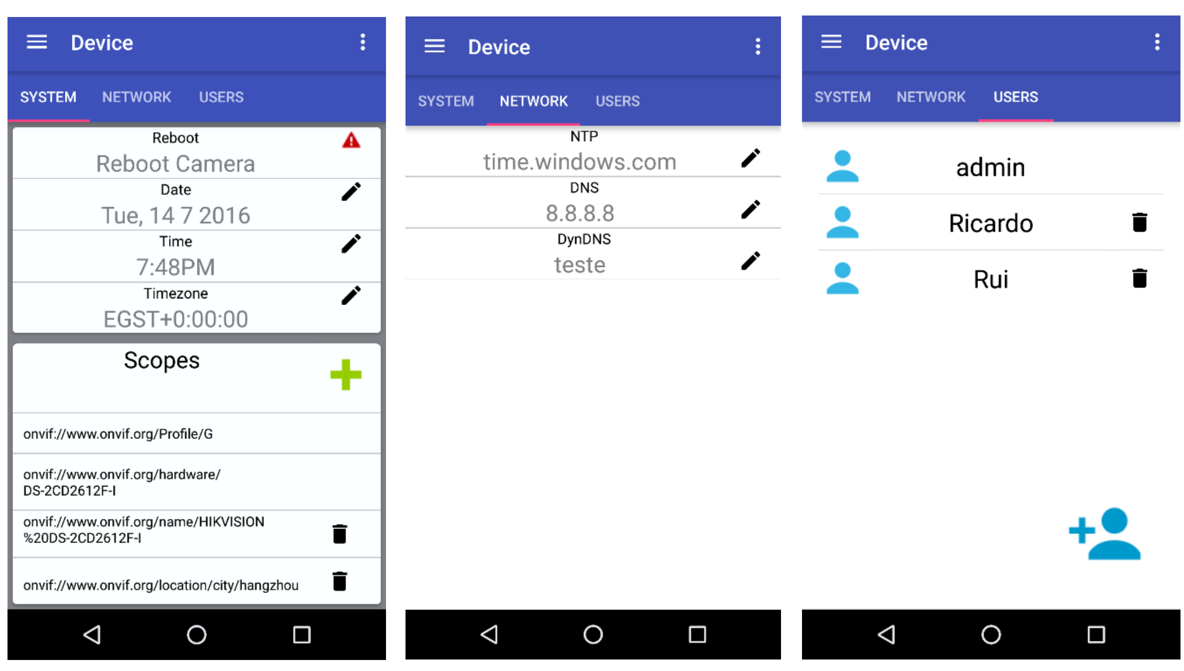


Figura X.0.7 - *Tab system* **Figura X.0.6** - *Tab network* **Figura X.0.5** - *Tab users*

A Toolbar tem opções para aceder às *capabilities* do Device como representado na Figura X.0.8.

Figura X.0.8 - Opções da Toolbar do serviço Device

Na Figura X.0.10, Figura X.0.9, Figura X.0.11 e Figura X.0.12 estão os Dialogs das *capabilities* para o serviço Device Managment. As *miscellaneous capabilities* são opcionais, pelo que, é emitida uma mensagem de aviso como na Figura X.0.12. Os campos do tipo *boolean* representam-se pelos símbolos certo ou errado seguidos do texto descritivo.

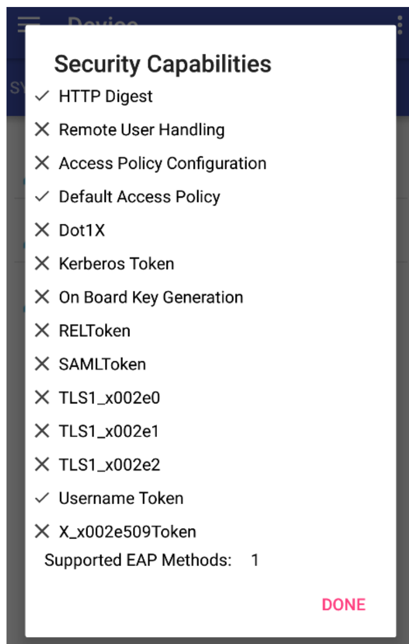


Figura X.0.9 - Dialog das *security capabilities*

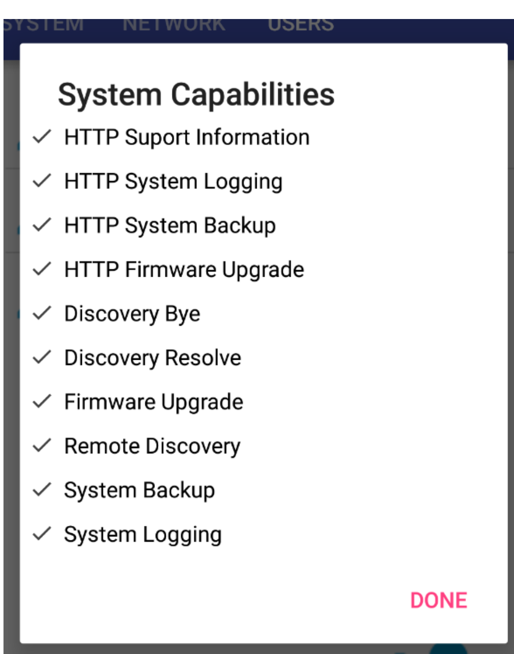


Figura X.0.10 – Dialog das *system capabilities*

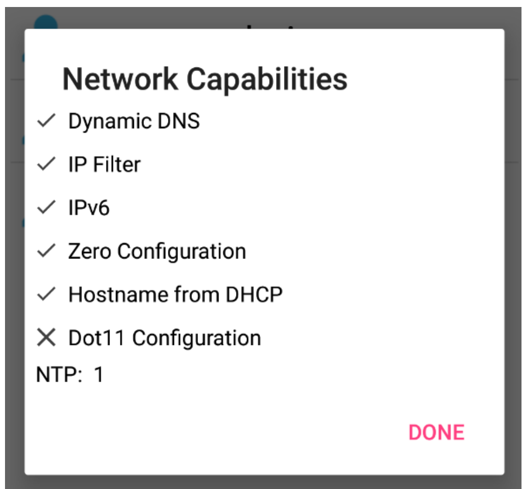


Figura X.0.11 - Dialog das *network capabilities*

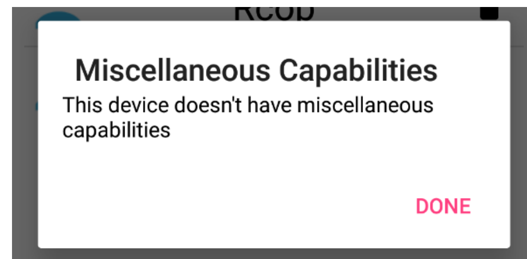


Figura X.0.12 - Dialog das *miscellaneous capabilities*

Menu DeviceIO

Adicionou-se as *capabilities* de Device IO ao menu para corresponder ao respetivo serviço. As *capabilities* do serviço Device IO são exibidas num Dialog ilustrado na Figura X.0.13 onde são exibidas informações sobre a quantidade de *digital inputs, serial ports, realy outputs, audio outputs/sources e video sources/outputs*. O aspeto do fragmento deste serviço está ilustrado na Figura X.0.14.

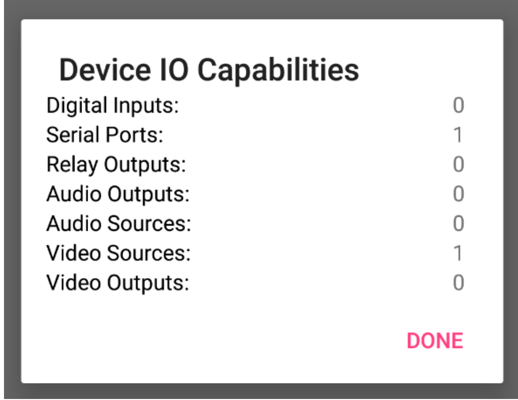


Figura X.0.13 - *Capablities* do serviço DeviceIO

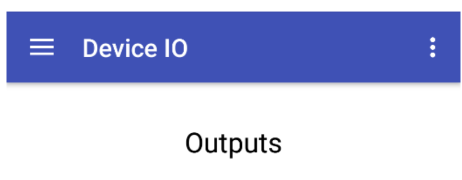


Figura X.0.14 - Fragmento DeviceIO

A nível de UI manteve-se o layout fazendo-se apenas a troca do tipo de comunicação. Neste layout são mostrados os Outputs existentes no dispositivo se existirem.

Menu Media

Adicionou-se a *capabilities* de Media ao menu do respetivo serviço. As *capabilities* do serviço media são exibidas num Dialog ilustrado na Figura X.0.15 onde são exibidas informações sobre o número máximo de perfis e suporte de rotação, *snapshot* e de *streaming*.

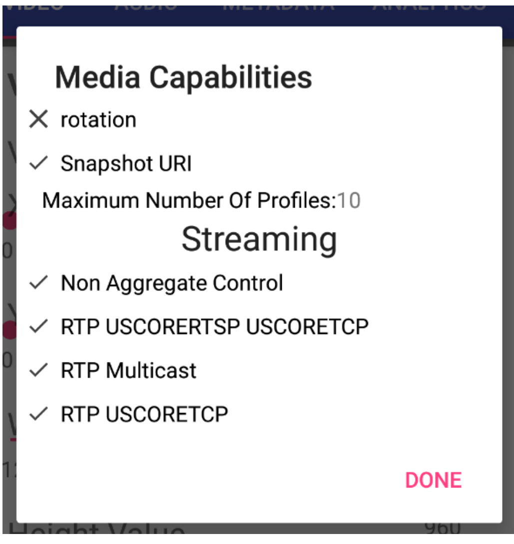


Figura X.0.15 - *Capabilities* do serviço Media

Os pedidos do serviço de Media requerem um campo para o *profile* sendo que é preciso uma opção de seleção dos mesmos. A seleção de um *profile* é feita através de uma partição abaixo da Toolbar que contém o nome do *profile* atual e um ícone a representar a ação de troca de *profiles*. Na troca de *profile* é utilizado um Dialog com a lista de *profiles* para seleção ilustrado na Figura X.0.16.

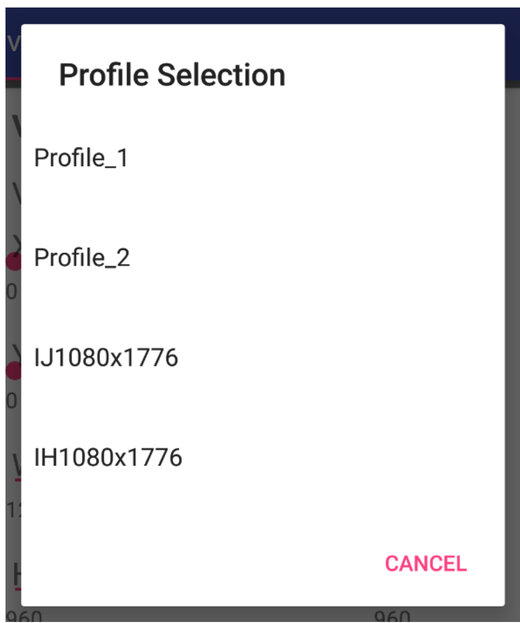
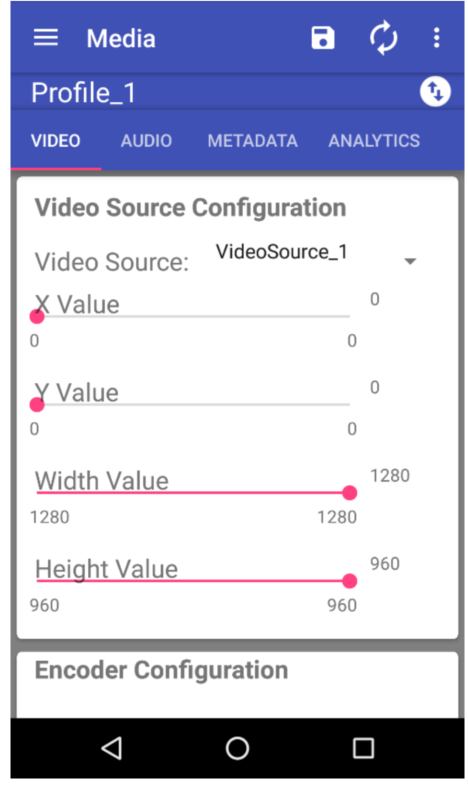


Figura X.0.16 - Dialog de seleção de um *profile*

Os dados de *media* estão distribuídos em 5 categorias distintas sendo necessário um tabulador no menu para Video/Audio/Metadata/Analytics/PTZ que permitem consultar e alterar as configurações de vídeo/áudio/*metadata/analytics/*PTZ respetivamente para o *profile* selecionado. Antes entrar no fragmento do serviço é efetuado o pedido para obter os *profiles* para seleção. São ainda carregadas as *options* e *settings* de *media* para apresentar os dados da câmara e as restrições de configuração dos mesmos. A aplicação de Media existente estava bastante completa, pelo que, trocou-se o tipo de comunicação sem a necessidade de fazer pedidos extra. A nível de UI haviam incoerências na forma de apresentar os dados e alguns *bugs,* sendo os mesmos, corrigidos. Reparou-se que os dados da UI que eram opcionais e não implementados pelo dispositivo estavam a ser apresentados sem qualquer tipo de valor existente. Desta forma, aplicou-se a lógica de esconder as Views para os dados opcionais que não fossem suportados pelo dispositivo, garantindo que o utilizador não seja induzido em erro ao configurar as definições de *media.*

Os dados estão organizados por cartões em todos os tabuladores porque todas as categorias de *media* têm dados distintos. Assim, caso existam dados suportados pelo dispositivo de uma dada categoria são apresentados em cartões como na Figura X.0.17, caso não haja qualquer tipo de informação os cartões desaparecem como na Figura X.0.18. O utilizador só visualiza os cartões que

têm qualquer tipo de informação suportada pelo dispositivo restringindo corretamente as configurações possíveis.



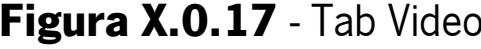


Figura X.0.17 - Tab Video

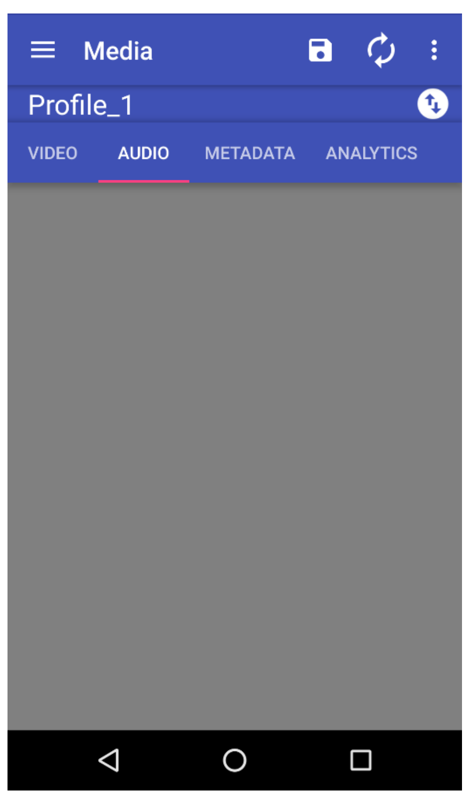


Figura X.0.18 - Tab Audio sem cartões suportados

Os tipos de cartões do tabulador Video e respetivos dados são listados na Tabela X.0.1:

Tabela X.0.1 - Dados do tabulador Vide de Media

Video Source Configuration	Encoder Configuration	Multicast Configuration
Video Source	Encoder Type	Port
[X,Y]	Video Resolution	TTL
[Width, Height]	Quality	IP e tipo de IP
	Frame Rate Limit	Auto Start
	Encoding Interval	
	Bit Rate Limit	
	Session Timeout	
	Enconder Profile	
	GOV Length	

Os tipos de cartões do tabulador Audio e respetivos dados são listados na Tabela X.0.2:

Tabela X.0.2 - Dados do tabulador Audio de Media

Encoder Configuration	Multicast Configuration	Audio Source Configuration
Enconding	Port	Tokens Available
Sample Rate	TTL	
Bit Rate	IP e tipo de IP	
Session Timeout	Auto Start	

Os tipos de cartões do tabulador Metadata e respetivos dados são listados na Tabela X.0.3.

Tabela X.0.3 - Dados do tabulador Metadata de Media

Event Subscription	PTZ	Analytics	Multicast Configuration
Attributtes	Status		Port
Filter Type	Position		TTL
Subscription			IP e tipo de IP
			Auto Start

Os tipos de cartões do tabulador Analytics e respetivos dados são listados na Tabela X.0.4.

Tabela X.0.4 - Dados do tabulador Analytics de Media

Analytics Engine	Analytics Module
Engine Configuration Extension	Name
Atribute	Type
	Parameters
	Simple Item
	Element Item
	Extension

Os tipos de cartões do tabulador PTZ e respetivos dados são listados na Tabela X.0.5.

Tabela X.0.5 - Dados do tabulador PTZ de Media

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

Menu Imaging

Adicionou-se as *capabilities* de Imaging ao menu do respetivo serviço. As *capabilities* do serviço Imgaing são exibidas num Dialog ilustrado na Figura X.0.19. O único campo revelado é do tipo *boolean* sobre a *image stabilization*.

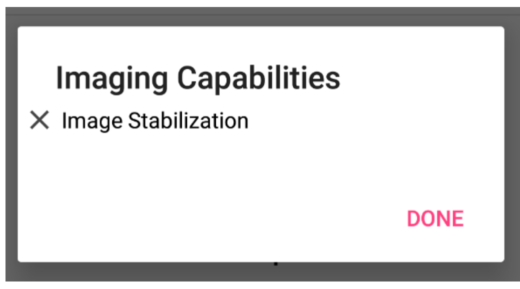


Figura X.0.19 - *Capabilities* do serviço *imaging*

Os pedidos do serviço *imaging* necessitam de um campo para o *video source* pretendido de modo a apresentar as configurações de cada um deles. Deste modo é preciso uma opção de seleção de um *video source* para se conseguir fazer os pedidos necessários. A seleção de um *video source* é feita através de uma partição abaixo da Toolbar que contém o nome do *video source* atual e um ícone a representar a ação de troca do mesmo. Quando se pretende trocar de *video source* é utilizado um Dialog com a lista dos *video sources* para seleção como na Figura X.0.20.

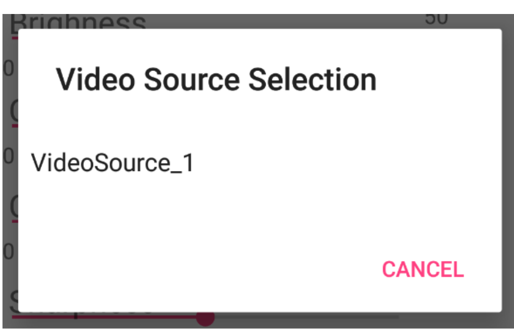


Figura X.0.20 - Dialog de seleção do *video source*

As funcionalidades de *imaging* fornecidas pela aplicação existente permitia consultar o estado e opções de configuração de *imaging*, pelo que, faz mais sentido apresentar os valores reais das configurações restringidas pelas opções de forma a permitir configurar os valores possíveis. Sendo assim, obtem-se os valores dos parâmetros de *imaging* configurados na câmara e usa-se as opções para restringir a configuração de valores válidos. Para os campos opcionais é retirada a View dos parâmetros caso não seja implementado. Removeu-se os fragmentos de consulta das opções e fez-se a adição de um fragmento com os valores configurados no dispositivo e que permite fazer a alteração dos mesmos respeitando as opções de *imaging* para o respetivo dispositivo. As propriedades de *imaging* de um *vídeo source* são divididas em cartões de acordo com as categorias distintas. Na Figura X.0.21 identifica-se abaixo da Toolbar o local de troca de *video sources* com a designação do selecionado, os cartões com os campos dos *settings* e as opções da Toolbar que permitem guardar alterações e ver as *capabilities*. Existe um ícone para ver a *preview* de um *snapshot* que ajuda a perceber o impacto das configurações.

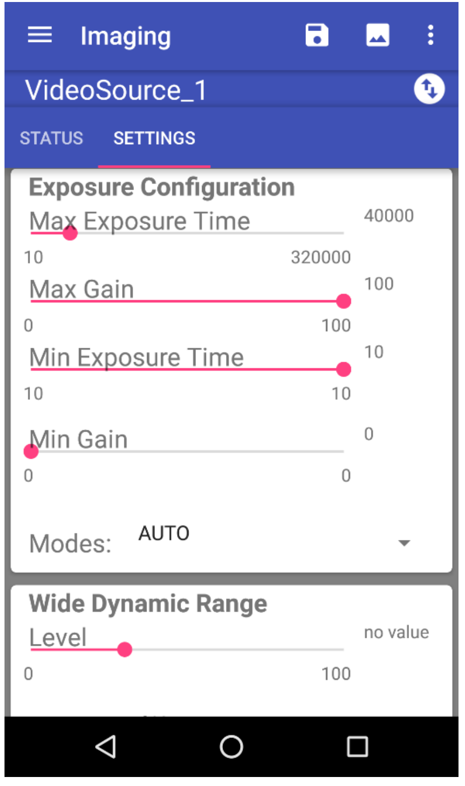


Figura X.0.21 - Tab settings do menu imaging

Os tipos de cartões do tabulador Settings e respetivos dados são listados na Tabela X.0.6.

Tabela X.0.6 – Dados do tabulador ImageSettings de Imaging

Settings	**Exposure Configuration**	**Focus**	**Wide Dynamic Range**	**Backlight Compensation**	**White Balance**
Brighness	Max. Exposure Time	Auto Focus	Level	Level	White Balance
Color Saturation	Max. Gain	Mode	Modes	Modes	ybGain
Contrast	Max. Iris	Default			yrGain
Sharpness	Min. Exposure Time	Speed			
Ir-cutFilter Modes	Min. Gain	Far Limit			
	Min. Iris	Near Limit			
	Modes				
	Exposure Time				
	Gain				
	Iris				
	Priority				

Menu PTZ

As funcionalidades PTZ são úteis na reprodução da *stream* de vídeo para fazer *pan, tilt e zoom.* No entanto, é possível consultar informações sobre as *capabilities* e os *presets* dos perfis de *media*, justificando assim, um menu para gestão das opções PTZ. Um fragmento é suficiente para englobar todas estas funcionalidades, pois, a maioria dos pedidos são usados quando se reproduz um vídeo em que é necessário saber quais as opções e permitir os movimentos suportados. As *capabilities* deste serviço consistem nos parâmetros *eFlip* e *reverse* como ilustrado na Figura X.0.22.

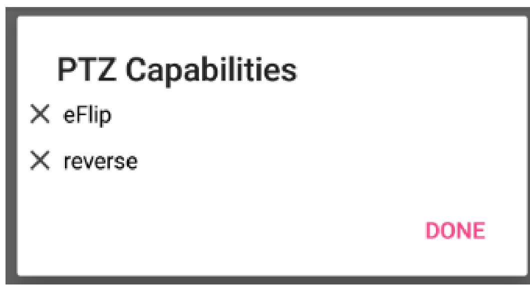


Figura X.0.22 – Dialog das PTZ *capabilities*

A câmara disponibilizada para este trabalho não suporta PTZ pelo que foi um dos serviços com menos prioridade de desenvolvimento na aplicação, uma vez que, não se pode realizar os testes desejados. Este menu é uma adição às aplicações existentes e deve conter informações de configurações e de consulta para o utilizador sobre as funcionalidades PTZ. Quando um dispositivo não suporta PTZ é exibido um Dialog com um aviso como na Figura X.0.23.

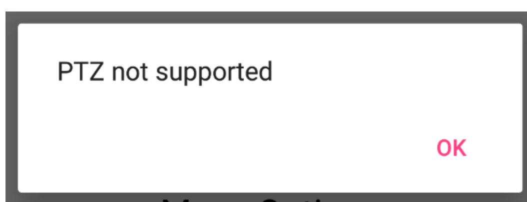


Figura X.0.23 - Dialog com mensagem de PTZ não suportado

Menu Analytics e Events

O serviço de Analytics e Events estão por desenvolver no servidor. Desta forma estes serviços aparecem no painel lateral de navegação, mas à entrada do serviço é apresentada uma mensagem a explicar que o serviço ainda não está implementado como na Figura X.0.24. Futuramente deve-se pôr as respetivas informações sobre estes serviços nos respetivos locais, sendo que, pode haver algumas modificações que podem advir da integração destes serviços.

Figura X.0.24 - Dialog com mensagem de serviço não implementado

Menu Stream

O menu de Stream tem como funcionalidade base mostrar o vídeo da câmara. A *stream* utiliza o protocolo RTSP, cujo endereço, é obtido através do pedido getStreamURI. Para mostrar a *stream* é utilizada uma atividade com visualização horizontal. Um exemplo de visualização da *stream* de vídeo é exibido na Figura X.0.25.

Figura X.0.25 – Activity do menu Stream.

Painel de Navegação Lateral

O painel de navegação lateral ilustrado na Figura X.0.26, tem 8 opções, uma para reproduzir a *stream* de vídeo e as restantes para aceder aos serviços. Este painel pode ser acedido através de um *swipe* para a direita começado através do limite do canto esquerdo do ecrã, tendo como alternativa, o típico botão com o símbolo do hambúrguer presente na Toolbar. Para fechar o painel utiliza-se o botão em forma de seta inversa, *swipe* para a esquerda ou um *tap* fora do painel. Quando se seleciona uma opção é criado um novo fragmento e fechado o painel de navegação. A funcionalidade de troca de câmaras neste painel foi descontinuada, uma vez que, é mais rápido aceder ao menu principal e selecionar a câmara pretendida do que abrir o painel, abrir a lista de câmaras e selecionar a câmara pretendida. Além disso é usual nas aplicações Android a utilização do *back button* para navegar para a atividade anterior presente na *stack* de atividades.

Em conclusão o painel de navegação lateral tem a função de fornecer acesso aos fragmentos de serviço e à atividade de Stream. Quando se acede a uma opção do painel é iniciado o novo fragmento/atividade e o painel é ocultado novamente.

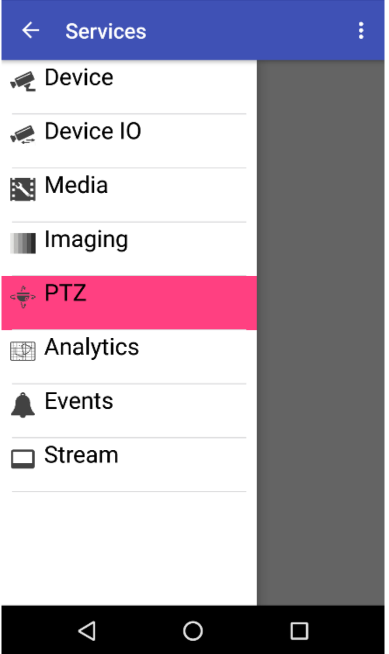


Figura X.0.26 - Painel de navegação lateral