Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

Bruno Simões Cruz

Evolução da Telefonia na Web

Web Telephony Evolution

Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

Bruno Simões Cruz

Evolução da Telefonia na Web

Web Telephony Evolution

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Doutor João Paulo Silva Barraca, Professor assistente convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor José Manuel Matos Moreira
Professor auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor João Paulo Barraca
Professor assistente convidado da Universidade de Aveiro

Profª Maria João Mesquita Rodrigues da Cunha Nicolau Pinto
Professora auxiliar do Departamento de Sistemas de Informação da Escola de Engenharia da Universidade do Minho

agradecimentos / acknowledgements

Desde já, gostaria de agradecer ao meu orientador, o Prof. Doutor João Paulo Silva Barraca, por todo o apoio e conselhos ao longo deste último ano. Gostaria também de agradecer ao Alexandre Brito pela ajuda e paciência, pelo conhecimento partilhado e simpatia. Um bem-haja ao Nuno Costa, pela visão, experiência e energia, que tanto me ensinaram. Ao Filipe Pinheiro, um obrigado por todo o apoio e flexibilidade. À WIT Software e a todos os seus elementos, nomeadamente o André Silva, equipa de WebRTC e o Nuno Campos, pela oportunidade, pela maneira como me receberam e todo o esforço em me ajudar.

Não posso deixar de agradecer a todos aqueles que me ajudaram durante o meu percurso académico, com especial apreço ao Prof. Cláudio Teixeira. Dedico este trabalho a toda a minha família, à minha paciente namorada e amigos, por todo o carinho e força, por estarem sempre lá. Mas, especialmente, dedico-o aos que já não consigo ver.

A todos, saibam que vos agradeço por tudo.

palavras-chave IMS, VoLTE, SIP, Mobicents, OneAPI, GSMA, OMA, RCS, WebRTC, API, integração de serviços

resumo Com a ameaça imposta às operadoras por aplicações OTT como WhatsApp ou Skype, diversas iniciativas coordenadas pela GSMA foram criadas para tentar responder a este fenómeno. Paralelamente, com a evolução de tecnologias como HTML5 e WebRTC, novos serviços como o Twilio têm surgido, oferecendo APIs para o desenvolvimento de novas aplicações Web. No entanto, a integração destas tecnologias em tradicionais redes de telecomunicações não faz parte das actuais especificações. Sendo assim, o objectivo desta dissertação consiste na especificação e implementação de um protótipo baseado nestas tecnologias emergentes, integrado com uma rede IMS. Primeiramente, foi feito um estudo do estado de arte, definindo requisitos e casos de uso a serem explorados. De seguida, o desenho da solução foi feito e implementado, tendo sido criado uma plataforma que alia WebRTC e a OneAPI da GSMA (que define funcionalidades básicas para operadores), oferecendo interoperabilidade entre ambos os mundos. A solução é composta por um servidor aplicacional que expõe a API e gateway WebRTC, tendo sido testada e considerada adaptada às necessidades estabelecidas.

keywords IMS, VoLTE, SIP, Mobicents, OneAPI, GSMA, OMA, RCS, WebRTC, API, service integration

abstract With the threat to operators by OTT applications such as Skype or WhatsApp, several initiatives coordinated by GSMA were created in an effort to respond to this phenomenon. In parallel, with the evolution of technologies such as HTML5 and WebRTC, new services such as Twilio are now available, offering APIs for web application development. However, the integration of these technologies and traditional telecommunication networks is not a part of the current standards. As such, the objective of this dissertation is the specification and implementation of a prototype based on these emerging technologies, integrated in an IMS network. First, a state-of-the-art analysis was made, defining requirements and use-cases to be explored. Secondly, the design and implementation of the solution was done, creating a platform that unites WebRTC and GSMA's OneAPI (which exposes basic operator features), offering interoperability between both worlds. The solution is composed by an application server that exposes the API and a WebRTC gateway, having been successfully tested and adapted to the established needs.

Table of Contents

Index of Figures xvii
Index of Tables xix
Index of Snippets xxi
Glossary xxiii
Introduction 1
1.1. Motivation 2
1.2. Objectives 2
1.3. Contributions 3
1.4. Document structure 3
IMS centric communication 5
2.1. IP Multimedia Subsystem 5
2.1.1. Transport layer 7
2.1.2. Control or IMS layer 8
2.1.3. Service or Application layer 10
2.1.4. Example IMS flows 11
2.1.5. Voice over LTE 12
2.2. IMS core implementations 15
2.2.1. Clearwater 15
2.2.2. OSIMS – OpenIMS Core 17
2.2.3. Kamailio 18
2.2.4. Core comparison 19
2.3. Web oriented RTC technologies 19
2.3.1. WebRTC 19
2.3.2. Other approaches 21
2.3.3. Technology comparison 22
2.4. Web-oriented RTC on IMS 23
2.4.1. Existing solutions 26
2.5. Telephony APIs 27
2.5.1. Twilio 27
2.5.2. Plivo 28
2.5.3. OneAPI 29
2.5.4. Summary 30
2.6. Chapter summary 31
Requirements for a WebRTC integration on IMS 33
3.1. Example use-cases 34
3.2. API features 36
3.2.1. Notification mechanism 38
3.2.2. API analytics 39
3.2.3. Integration features 39
3.3. Portal features 40
3.3.1. User provisioning API 41
3.4. WebRTC features 42
3.5. Network interfaces 43
3.6. Non-functional features 44
3.6.1. Availability and scalability 44
3.6.2. Availability strategy 45
3.6.3. Performance 46
3.6.4. Interoperability 46
3.6.5. Security 47
3.7. Chapter summary 47
Solution proposed 49
4.1. Application server 50
4.1.1. OneAPI layer 53
4.1.2. OneAPI notification manager 54

4.1.3. Database Manager 55
4.1.4. SIP Servlet 56
4.1.5. Authentication flow 57
4.1.6. Call transfer flow 58
4.1.7. Call hold/resume flow 58
4.1.8. Messaging flow 59
4.1.9. Notification flow 60
4.2. Portal 61
4.2.1. User Provisioning API 63
4.2.2. Developer registration 63
4.2.3. Application submission and approval 64
4.2.4. Application suspension 65
4.3. WebRTC 66
4.3.1. OpenIMS modifications 66
4.3.2. User interface 67
4.4. Third party application 69
4.5. Chapter summary 70
Evaluation and results 73
5.1. Testbed description 74
5.2. Call throughput 75
5.2.1. IMS signalling 75
5.2.2. IMS signalling and RTP exchange 80
5.2.3. AS 84
5.2.4. WebRTC and AS 87
5.2.5. Test results summary 89
5.3. Mouth-to-ear delay 89
5.3.1. Peer-to-peer call 91
5.3.2. RTPproxy call 92
5.3.3. AS call 93
5.3.4. WebRTC calls 93
5.3.5. GSM calls 95
5.3.6. Test results summary 97
5.4. Chapter summary 98
Conclusion 99
6.1. Future work 100
References 103
A) Requirements 109
A.1) Functional requirements 109
A.2.1) API/WebRTC requirements 109
A.2.2) Portal requirements 114
A.3) General requirements 118
B) OneAPI layer 120
B.1) Authentication deviation 120
B.2) Features 121
B.2.1) User authentication 121
B.2.2) Sending an SMS 122
B.2.3) Sending an MMS 124
B.2.4) Notifications 128
B.2.5) Make Call 129
B.2.6) Retrieve Call Information 132
B.2.7) End Call 133
B.2.8) Add To Call 135
B.2.9) Retrieve Participant Information 136
B.2.10) Delete Participant from Call 137
B.2.11) Hold/Resume Call 138
C) User Provisioning API 141
C.1) Features 141

C.1.1) Request authentication 141
C.1.2) User creation/update 141
C.1.3) User deletion 145
C.1.4) HTTP responses 147
D) User provisioning solutions 148
D.1) Welcome notice 148
D.2) Third party registration 149
D.3) Diameter/Cx integration 149
E) Notification mechanisms solutions 151
E.1) Push alternatives - WebSockets/HTTP Streaming 151
E.2) Pull alternatives - Long polling 151
F) Testbed components and software 152
G) OpenIMS modification 153
H) MTS default proprieties 153
I) P-CSCF error and modification 154

Index of Figures

Figure 1 - IMS architecture 6
Figure 2 - Closer view on control layer components 9
Figure 3 - User registration flow within an IMS core 11
Figure 4 - Call invitation flow within an IMS core 12
Figure 5- EPC, LTE and IMS relation 13
Figure 6 - Call invitation flow within an LTE/IMS network 14
Figure 7 - Clearwater's architecture (with Bono) 15
Figure 8 - Blind call transfer example 16
Figure 9 - OpenIMS's architecture 17
Figure 10 - Kamailio's IMS capabilities 18
Figure 11 - WebRTC's architecture 20
Figure 12 - SBC RTC on IMS solution 23
Figure 13 - Gateway architecture for RTC on IMS 24
Figure 14 - Click-to-Call flow diagram (initial invite exchange) 24
Figure 15 - Plivo's high-level architecture 28
Figure 16 - Logical view of the system 33
Figure 17 – Calling a client from a CRM application 35
Figure 18 – Adding a friend to an on-going call 35
Figure 19 – Third-party call example 37
Figure 20 – Webhook notification mechanism 38
Figure 21 - Apigee as a proxy 39
Figure 22 - User provisioning API 42
Figure 23 - Network load-balancing 45
Figure 24 – WTE reference architecture 49
Figure 25 - VaaS' call triggering workflow 51
Figure 26 – VaaS' modules 51
Figure 27 – WTE's AS new and modified modules 53
Figure 28 - OneAPI layer methods 54
Figure 29 - OneAPI Notification Manager methods 55
Figure 30 - Database manager methods 56
Figure 31 - SIP Servlet methods 56
Figure 32 - User/Application authentication flow 57
Figure 33 - Call transfer flow 58
Figure 34 - Call Hold flow 59
Figure 35 – Pager-mode message flow example 60
Figure 36 - Webhook notification flow 60
Figure 37 - Long-polling notification flow 61
Figure 38 - Portal's screenshot 61
Figure 39 - WTE Portal's components 62
Figure 40 - Developer registration (user side) 64
Figure 41 - Developer registration (administration side) 64
Figure 42 - Application publishing request 65
Figure 43 - Application suspension flow 65
Figure 44 - WWC chat interface 67
Figure 45 - Contact creation event on the WWC 69
Figure 46 - Third party application architecture 69
Figure 47 - Default script steps and signalling flow 76
Figure 48 - Call establishment times at 165 SC (signalling only) 77
Figure 49 - Call establishment times at 330 CS (signalling only) 77
Figure 50 - Call establishment times at 495 SC (signalling only) 77
Figure 51 - Call establishment times at 660 SC (signalling only) 78
Figure 52 - Call establishment time distribution, in seconds (330 SC, signalling only) 79
Figure 53 - Call establishment time distribution, in seconds (660 SC, signalling only) 79

Figure 54 - RTPproxy signalling flow....80
Figure 55 - Call establishment times at 66 SC (signalling and RTP)....81
Figure 56 - Call establishment times at 99 SC (signalling and RTP)....81
Figure 57 - Call establishment times at 132 SC (signalling and RTP)....82
Figure 58 - Call establishment times at 165 SC (signalling and RTP)....82
Figure 59 - MTS response time, in seconds, at 99 SC....83
Figure 60 - MTS response time, in seconds, at 132 SC....83
Figure 61 - Call establishment time distribution, in seconds (99 SC, signalling and RTP)....84
Figure 62 - API signalling flow....85
Figure 63 - MTS response time, in seconds, at 99 SC....85
Figure 64 - MTS response time, in seconds, at 132 SC....86
Figure 65 - WebRTC load-test approach....88
Figure 66 - MTS response times (20 concurrent calls)....88
Figure 67 – Mouth-to-ear test configuration....90
Figure 68 - Peer-to-peer test configuration....91
Figure 69 - Peer-to-peer audio comparison....91
Figure 70 - RTPproxy test configuration....92
Figure 71 - RTPproxy audio comparison....93
Figure 72 - AS audio comparison....93
Figure 73 - WWC audio comparison....94
Figure 74 - AS/WWC test configuration....94
Figure 75 - WWC/AS audio comparison....95
Figure 76 - GSM test configuration....95
Figure 77 - GSM delay test result (same operator)....96
Figure 78 - GSM delay test result (different operators)....96
Figure 79 - WCDMA delay test result (same operator)....97
Figure 80 - Messaging flow....124
Figure 81 - Call Triggering flow....132
Figure 82 - Hang-up flow....134
Figure 83 - User creation flow....145
Figure 84 - Welcome notice....148
Figure 85 - Third party registration....149
Figure 86 – Diameter/Cx integration....150

Index of Tables

Table 1 - Feature comparison 30
Table 2 - Performance test results summary 89
Table 3 - Mouth-to-ear test results summary 97
Table 4 - Application authentication functions 110
Table 5 - Developer authentication functions 110
Table 6 - User authentication functions 110
Table 7 - Call operations overview 112
Table 8 - Messaging functions 112
Table 9 - Notification functions 113
Table 10 - Presence functions 114
Table 11 - Available analytics functions 114
Table 12 - Billing function 115
Table 13 - Administrative actions 115
Table 14 - Marketplace functions 116
Table 15 - Application management operations 116
Table 16 - Developer management operations 117
Table 17 - User actions 117
Table 18 - User management operations 118
Table 19 - General requirements 119
Table 20 - HTTP response codes 147
Table 21 – MTS default configuration changes 153

Index of Snippets

Snippet 1 – SIP message format....8
Snippet 2 - Example user creation body....63
Snippet 3 - Call logging in the AS....86
Snippet 4 - Example resource showing the application token location....121
Snippet 5 – Example user authentication request body....121
Snippet 6 – Example user authentication request....122
Snippet 7 - Example user authentication response....122
Snippet 8 - Example SMS send request body....123
Snippet 9 - Example SMS send request....123
Snippet 10 - Example SMS send response....123
Snippet 11 - Example MMS send request body (part 1)....125
Snippet 12 - Example MMS send request body (part 2)....126
Snippet 13 - Example MMS send request....127
Snippet 14 - Example MMS send response....128
Snippet 15 - Example notification request....129
Snippet 16 - Example notification response....129
Snippet 17 - Example call triggering request body (basic call session)....130
Snippet 18 - Example call triggering request body (complete call session)....130
Snippet 19 - Example call triggering request....131
Snippet 20 - Example call triggering response....131
Snippet 21 - Example call information retrieval request....132
Snippet 22 - Example call information retrieval response....133
Snippet 23 - Example call ending request....133
Snippet 24 - Example call ending response....134
Snippet 25 - Example add to call request body....135
Snippet 26 - Example add to call request....135
Snippet 27 - Example add to call response....136
Snippet 28 - Example participant information retrival request....137
Snippet 29 - Example participant information retrieval response....137
Snippet 30 - Example participant deletion from call request....138
Snippet 31 - Example participant deletion from call response....138
Snippet 32 - Example hold call request body....139
Snippet 33 - Example hold call request....139
Snippet 34 - Example hold call response....139
Snippet 35 - Example user creation request body (JSON)....142
Snippet 36 - Example user creation request body (XML)....142
Snippet 37 - Example user creation request (with JSON body)....143
Snippet 38 - Example user creation request (with XML body)....143
Snippet 39 - Example user creation response....144
Snippet 40 - Example user deletion request body (JSON)....145
Snippet 41 - Example user deletion request body (XML)....146
Snippet 42 - Example user deletion request (with JSON body)....146
Snippet 43 - Example user deletion request (with XML body)....146
Snippet 44 - Example user deletion response....147
Snippet 45 - OpenIMS modification for evaluation....153
Snippet 46 - OpenIMS's SDP utilities module....154
Snippet 47 - SDP utilities modification....154

Glossary

API	Application Programming Interface, defines specific means of interaction between software components.
AS	Application Server, servers that host and execute services made available to network's subscribers.
B2BUA	Back-to-Back User Agent, a type of application server architecture, designed to enable manipulation of individual communication legs.
CDR	Call Detail Record, the name of the logs generated to describe a given call, using details such as participants, duration, error messages, among others.
Codec	Short for COmpressor/DECompressor, a mechanism of encoding or decoding data streams for faster media exchange.
CPS	Calls Per Second, an aggregate metric of performance used to indicate the number of new calls per second that a telephony server is able to handle.
CSCF	Call Session Control Function, nodes that manage the signalling from users to services and from one network to another, composing the main components of an IMS core.
DTMF	Dual-Tone Multi-Frequency signalling, the tones generated by pressing a key in a telephone's keypad. The capture of these signals is crucial to automated systems in call centers, for example.
EPC	Evolved Packet Core, the core of an LTE network, introduced by 3GPP.
GSMA	Groupe Speciale Mobile Association, a group of mobile operators motivating the standardization of GSM networks.
HSS	Home Subscriber Server, the centralized database for an IMS network containing user-related information and exposing methods for authentication and authorization.
HTTP	Hypertext Transfer Protocol, a communications protocol widely used in the Internet.
IMS	IP Multimedia Subsystem. A novel network architecture designed for communication convergence and easing the creation and deployment of multimedia services - one could see it as Service Oriented Architecture (SOA) for telecommunication usage.
IP	Internet Protocol, one of the main protocols used for establishing Internet communications.
IP PBX	Internet Protocol Private Branch eXchange, a system that connects private extensions to public networks, over an internet protocol stack.
KPI	Key Performance Indicators, metrics used to measure performance across a system.
LTE	Long Term Evolution. A novel standard of high-speed, wireless communications destined for mobile terminals.
MMS	Multimedia Messaging Service. Enables the sharing of multimedia content via what would otherwise be a simple text message.

MSRP	Message Session Relay Protocol, a protocol responsible for exchanging the content of a messaging session between participants.
OMA	Open Mobile Alliance, responsible for delivering technical application and service framework specifications.
PoC	Proof of Concept, a demonstration of the feasibility of a project through, in this case, the development of an example application.
PSTN	Public Switched Telephone Network, the aggregate of circuit-switched, public telephone networks.
QoS	Quality of Service, general service quality evaluation, mainly focused on its user's perspective.
RCS	Rich Communication Services, a service designed to offer enhanced communication with features such as presence tracking (as one would see in a chat service) and easy file-sharing.
RTC	Real-Time Communication, the concept referring to communications done in a near instant manner.
RTP	Real-Time Transfer Protocol is a network protocol used in real-time applications.
SBC	Session Border Controller, an IMS component responsible for security measures and signalling control, among other functionalities.
SDP	Session Description Protocol, the protocol that describes the media initialization parameters, used for session management and parameter negotiation (what the clients will be using for communication purposes).
SIP	Session Initiation Protocol, a protocol responsible for signalling and managing communications between all the participants of a certain session, used wieldy in VoIP services and one of the most used protocols in IMS networks.
SMS	Short Message Service, a text message service best known for its mobile usage.
TAS	Telephony Application Server, a B2BUA server designed to expose call-related services (from call setup to call forwarding features).
VoIP	Voice over Internet Protocol. The concept describing the technologies designed to enable audio communication via the Internet.
VoLTE	Voice over LTE. A concept similar to VoIP but adapted to LTE-supporting mobile networks.
WebRTC	Web Real-Time Communication, an API for voice/video communication and data transfer between browsers.
3GPP	3rd Generation Partnership Project, a project formed by the collaboration between major standard defining organizations in the telecommunications realm, focusing on the definition of standards that will be used in emerging technologies.

Chapter 1

Introduction

From landline phones and legacy networks to softphones and telephony over the Internet, telecommunications have come a long way over these past years, changing the way people interact with each other. High-speed Internet access has only recently become available over mobile networks, with users becoming more frequent consumers of mobile data plans. The evolution of smartphones, with more processing power and more user-focused features, as brought an ever-increasing interest in mobile applications, with the creation of marketplaces designed to facilitate their discovery. With the merging of these worlds, both mobile and web, a multitude of telephony applications have become available, seen by users as cheaper alternatives to communicate across the world. Applications such as Skype, Line or WhatsApp give the users the power to call each other in real-time over the internet, lowering the costs these calls would mean to the parties involved. These so called Over-the-Top (OTT) applications, conveying only financial gain to the companies behind them, are turning operators into mere bit pipes: in this scenario, they only assure the connectivity to the Internet and charge the traffic used, but gain nothing over the multimedia communication established. This constitutes a major threat to operators, who see their profits decrease with the increase in usage of these types of applications. With the evolution of standards and with associations such as GSMA (Groupe Speciale Mobile Association, formed by several major telecom operators) motivating and empowering the searches for innovation, solutions are being proposed to attract consumers to the operator's world. From network architectures such as IMS (IP Multimedia Subsystem), designed to create a bridge between traditional telecom and the Internet, operators are considering new ideas to maintain profits and consumers, solutions such as the Rich Communication Services (RCS) program. Not only multimedia calls, the objective of these solutions is to give consumers the features they are already used to and most appreciate in the OTT world, features such as instant messaging, multimedia content sharing (either within a call or by messaging services) – enhanced experiences designed to offer rich, although simple means of communication through the operator's realm.

Parallel to telecoms, Internet communication is also in turmoil, with standards being proposed to enable new Real-Time Communication (RTC) experiences. Having started with Voice over Internet Protocol (VoIP) and Flash technologies (Flash being a multimedia platform that allows multimedia content exchange in real-time), advances are now being made on RTC to allow for grander experiences in this field, enabling communication without the need of specific

applications or browser plugins (something obligatory in Flash). This standard, known as WebRTC, aims to allow not only multimedia calls but the means to exchange data over secure connections and at the same easing the hassle of installing other components. Although still open, this standard as seen much adoption on the market, gaining operators attention as new solutions to create interoperability between traditional networking and this technology are investigated.

However, connecting a mobile caller to an Internet one can be challenging, even with the technology available. With this in mind, this dissertation has as objective the specification, design and implementation of a prototype that will enable third-party developers, without telecommunications background, to integrate advanced communication functionalities into their web applications, providing value-added features integrated with telecommunication operators.

1.1. **Motivation**

With architectural advances such as the 3rd Generation Partnership Project (3GPP) defined IMS architecture, along with recent and emerging technologies such as WebRTC, the Internet and telecommunication worlds are experiencing a convergence, with the added strength over the search for interoperability – an objective otherwise inexistent in the standards created. WebRTC itself, a still open-standard that is growing in popularity, has brought new life to IMS, motivating the creation of a diversity of novelty web-oriented multimedia services. These services are sustained by brand new use-cases such as rich collaboration and RCS clients, services growing in popularity that offer telecom users features that they are used to in other web applications.

Such advances, technologies and use-cases are the main drive for this dissertation, where the research made will direct the work towards a platform which is able to unite network users to developers with no background in communication. Although the search for interoperability between both Internet and telecom users has been fomented by these latest developments, the task of connecting both worlds is daunting and complex. This highly motives the creation of the platform and this dissertation, aiming to provide a solution for this problem, creating a bridge between users and the means to enable communication between traditional clients and WebRTC-supported ones.

1.2. **Objectives**

This dissertation has as objective the design and development of a platform that foments interoperability between endpoints placed in traditional telecommunication networks and web-oriented clients, enabling their communication through calling and messaging. The solution's

development will require several details, from implementation of the component exposing a standardized API to the addition of support for WebRTC. A complete platform will be created, featuring a user-centered approach towards service discovery, enabled by the creation of a portal that features a marketplace where network services will be published. Featuring an analysis on market state, this document may also be seen as a technical options research, exposing current trends and alternatives, such as IMS cores, RTC technologies, telephony APIs and alternative for the creation of the solution.

1.3. **Contributions**

This dissertation was developed in an enterprise environment, working with a telecommunications-geared company called WIT Software. A company renowned for its products in the telecommunication market, WIT offers a multitude of novel technological solutions from mobile applications to RCS clients and works with major operators worldwide. From this dissertation resulted several contributions to the business.

Firstly, the solution made is now part of WIT Software's offerings, having the developed proof-of-concept already been shown to a major client and considered an interesting solution to the problem at hand. Secondly, the work made on the WebRTC client that is used in this platform augmented the product itself, implementing features that had been planned previously but not yet developed. Thirdly, with the knowledge gained with this dissertation the author was able to help in a company-provided IMS/VoIP course, teaching WIT collaborators, with no knowledge in this field, technical insight into how these technologies behave when dealing with tasks from peer-to-peer calls to call forwarding and application server deployment.

1.4. **Document structure**

This dissertation will be divided in six chapters, with the first and last containing introduction and final conclusions, respectively:

- Chapter 2 – this chapter will present the state of the art for this dissertation, describing the research made in terms of IMS, RTC technologies, existing integration solutions and telephony APIs;

- Chapter 3 – this chapter will describe proposed requirements for an IMS and RTC integration, with considered use-cases being presented along with system requirements;

- Chapter 4 – in this chapter, development results of the proposed solution will be analyzed, covering what was implemented and how it was done, obstacles that arouse from the design decisions and limitations that have emerged;

- Chapter 5 – this chapter will feature solution evaluation, presenting testing results and discussing them, providing in-depth knowledge on the solution's behavior.

At the end of each, a summary will be made, resuming and emphasizing the most relevant topics and ideas. After the dissertation's conclusions, an annex section will be featured, composed of topics that provide further information on certain overviewed subjects.

Chapter 2

IMS centric communication

In order to pave the way towards a proper solution planning and to gain context on the fields and markets involved, where they are headed and what is being done, an investigation of the current state of the market regarding telephony API's, IMS cores and existing telephony over the web solutions was made.

As such, throughout this chapter, an introduction and proper historical view on the IMS network and RTC enablers (such as WebRTC or Flash) will be highlighted, in order to better understand the changes that these technologies have gone through these past few years. Furthermore, a research on the main components of such a solution must be made, as well as available products for web communication. Although not intended to provide a comprehensive list and description of the mentioned topics, an effort was made to obtain and present the most relevant solutions and concepts in the scope of this dissertation.

2.1. IP Multimedia Subsystem

From circuit switched networks (where an operator would route the call to its destination) to packet switched networks (where the Internet's own packet switching capabilities and IP usage enable communication between parties) and the concept of VoIP, innovation on the underlying means that enable people to connect and communicate with each other around the world is staggering and ever-evolving. Additionally, as communication drifted from fixed to mobile, came the cellular telephony expansion, where evolving standards from 3GPP and OMA (Open Mobile Alliance) brought a diversification of the services provided, from voice and multimedia messaging to e-mail and mobile Internet access, along with speed and reliability.

With the exploration of both wired and wireless telephony capabilities came the desire for a convergence of these technologies, where subscribers would be able to access Internet multimedia services from a diversity of devices (either fixed, home telephones or mobile smartphones). With this in mind, IMS was developed, leveraging these services with the existing networks and providing all functionalities needed for a "complete service delivery"[1] (from client registration to billing).

Not itself a technology but more precisely an architecture, IMS was first pushed in 2003, released by 3GPP for mobile usage, and has since seen its widespread use on mobile and fixed/IT networks[2]. With the increase of its popularity, especially among key players in the telecommunications market, IMS has become the industry standard for multimedia services, providing security, reliability and efficiency. From a market perspective, IMS lets telco companies drift from previous silo'ed services (i.e. services incapable of connection and operation with each other) and create advanced client-oriented services. This way, operators are able to roll-out their own applications in an easier manner[3], reducing operating costs and time-to-market. This makes them able to provide appealing new services, seizing new clients. By using these means, these telecommunication companies are also able to compete with popular Over-The-Top (OTT) communication applications (Skype, WhatsApp, among others)[4] which only convey financial benefit to the company that offers the application itself, not the one supporting the communication from a user's phone. IMS's architecture is composed of three main layers (disregarding the device layer), each with its own purpose and commonly implemented components – as seen in Figure 1[5].

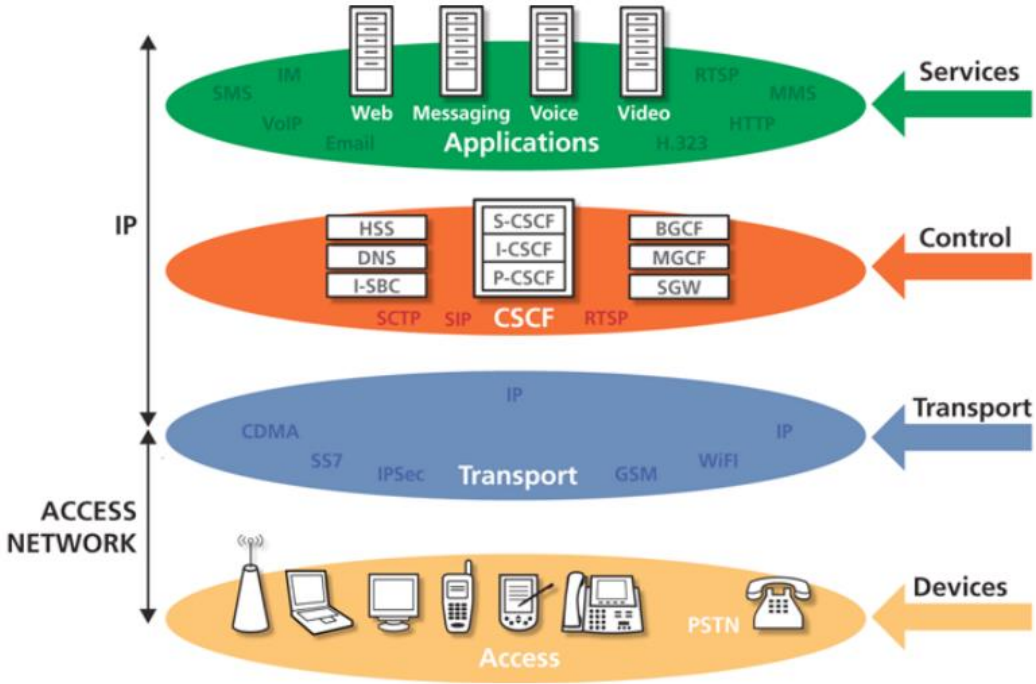

Figure 1 - IMS architecture

These layers will be covered in the next sections, an overview which is required for a better understanding of concepts presented throughout this dissertation, leaving aside low-level details such as their inner workings or implementation.

2.1.1. **Transport layer**

From softphones to computers, IMS enables users to connect to the network with a variety of devices. Landlines are also supported: the transport layer has the means for a Public Switched Telephone Network device (PSTN) to interact with IP based ones, giving the analog data a proper conversion to IP packets (in the PSTN gateway previously shown on Figure 1).

This layer is where these devices are able to interact with the network, using different connection types such as wireless internet, mobile networks or LTE and exchanging a signalling protocol known as SIP. SIP itself is a crucial protocol used within the network and indeed in the VoIP world. Specified in RFC 3261[6], it is one of the several protocols seen on the Service or Application layer of an IMS network, used for signalling and session management, enabling the control of call sessions, for example. It allows for three main session-related actions: call management, creation of calls between two or more participants and termination of those calls. From the exchange of SIP requests, one can negotiate what features will be available during the call session, the changing of these features amidst the session, user location and participant management.

In order to interact with the IMS network, User Agents (or UA, the names given to interacting entities) first need to go through a registration procedure, sending a SIP request called REGISTER to the network, with a certain expiration date which they must refresh, in order to keep themselves registered. After this action, and among the SIP commands available, one can find the commands to invite a peer to a call or terminate it (INVITE and BYE), cancel a sent invitation request or acknowledge a received invitation (CANCEL and ACK). SIP MESSAGE is also available, used when sending textual messages to other users, and, although there are many other commands, these six will be the most referred through this dissertation.

Similar to Hypertext Transfer Protocol (HTTP), a SIP request is composed by message headers and body – where one can find SDP. Whereas SIP is only used for signalling, to define and exchange session features the Session Description Protocol (SDP) is used. First published in 1998 on RFC 2327[7], SDP provides the means for a party to exchange specific session details and will normally be contained in the SIP message body. For instance, in SDP one can find the codecs that will be used and their priority, session owner and name and the time a session is active, where in SIP one finds contact identifiers and parties involved in a session (the “to” and “from” fields). An example of a SIP request for a call invite can be seen in Snippet 1.

```
INVITE sip:+351915661333@open-ims.test;user=phone SIP/2.0
Call-ID: 67597b0d43fb332a8a46eb054f27e2fa@127.0.0.1
CSeq: 1 INVITE
From:     <sip:wte@open-ims.test>;tag=72028314_2749a2ec_4fc1a424-7b5a-45af-8d56-
c4ddcad943b0
To: <sip:+351915661333@open-ims.test;user=phone>
Max-Forwards: 70
Contact: <sip:wte@127.0.0.1:5080>
Route: <sip:172.22.0.60:6060;orig;lr>
P-Asserted-Identity: <sip:+351915661333@open-ims.test>
Content-Type: application/sdp
Via:      SIP/2.0/UDP      127.0.0.1:5080;branch=z9hG4bK4fc1a424-7b5a-45af-8d56-
c4ddcad943b0_2749a2ec_6925275691444
Content-Length: 142

v=0
o=- 1400082303908 1400082303909 IN IP4 172.22.0.60
s=-
c=IN IP4 172.22.0.60
t=0 0
m=audio 15004 RTP/AVP 8
a=rtpmap:8 PCMA/8000
a=sendrecv
```

Snippet 1 – SIP message format

Depicted in Snippet 1 is an INVITE which contains information regarding from whom the call invite has been sent and to whom it was addressed (“From” and “To” field). Network-related information can also be seen, such as the IP in the contact address (“Contact” field) that locates the user or routing information such as the “Route” and “Via” fields, which show where the request will flow through. The SDP information shows that the only audio codec negotiated is G711 (defined by the acronym PCMA in the “a” field, which relates to session attributes) and that the RTP audio stream can be accessed in port 15004, with IP 172.22.0.60 (fields “m” and “c”, respectively).

2.1.2. **Control or IMS layer**

Represented in Figure 2[8], this layer is where SIP sessions are managed, being responsible for providing service/device communication, user authentication and more. As mentioned in the transport layer, SIP is used for signalling/session management. In order for the protocol to make its way through the network, different elements are required, such as the Call Session Control Functions (CSCF):

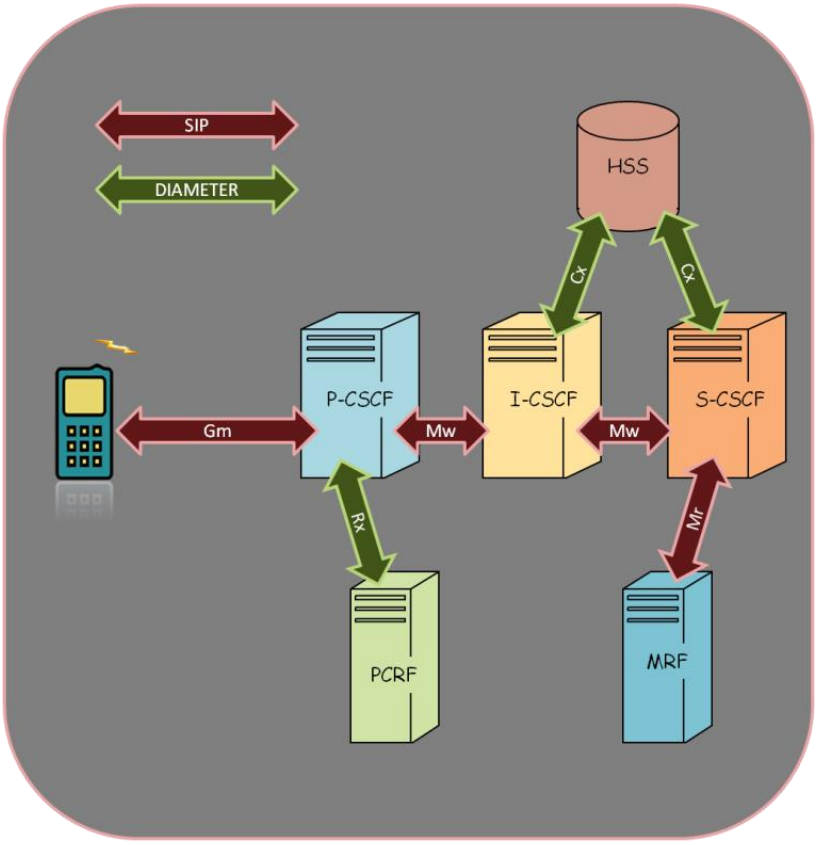

Figure 2 - Closer view on control layer components

- Proxy-CSCF - "(…) the first point of contact (…) to the user terminals"[9], it is the bridge between the converged networks and the IMS core, routing packets to the desired servers. It is also able to provide security functionalities (signal inspection, for example);
- Serving-CSCF - among other functions, performs session control/registration, maintains session states and routes traffic as needed to other network components or services[9];
- Interrogating-CSCF - acts as the "point of contact for peer IMS networks", facilitating the connections between the users and contacting the Home Subscriber Server (HSS) to determine user identification[9].

Alongside the CSCFs come other elements, such as the Home Subscriber Server (HSS), Media Resource Function (MRF) and Policy Control and Charging Rules Function (PCRF):

- HSS – the HSS is essentially a centralized database, storing the identities of all network users. Not only for session management, this element is very important for user authentication and authorization, in addition to maintaining control over interactions that certain users may have with application servers;

- MRF – this optional component is divided in two parts, one providing media controls – the Media Resource Function Controller (MRFC) – and the other providing processing functions - the Media Resource Function Processor (MRFP). These elements enable, for example, the playback of announcements and tones[9];
- PCRF – according to its standard definition, this is an "element that encompasses policy control decision and flow based charging control functionalities"[10]. Again, an optional element depending on required use-cases, this node acquires certain network responsibilities on billing and security policy up-keeping.

With this analysis, one can see how this layer encapsulates every detail regarding SIP signalling, from security issues to request forwarding. However, services that the network exposes are not included here, being left for the service layer.

2.1.3. **Service or Application layer**

In this top layer, one can find all Application Servers (AS) that have been deployed on the network. It is common to find multiple AS's, each with its own desired purpose and able to execute different multimedia services, such as:

- Presence Server – this server, through the management and tracking of the availability on subscribed users, allows a user to disclose his status (if online, busy, away or offline, for example) and to be notified of other user's status changes;
- Telephony Application Server (TAS) – this server provides call services, from call routing to conference bridging. It is also able to interact with the previously mentioned MRF to provide tones and announcements;

As mentioned, services are exposed by AS's deployed on the network. As such, one could use these capabilities to provide a custom AS, exposing an API for developer use and a portal for facilitating the creation of new applications. This scenario is directly related to the objective of this dissertation and one can see how this layer is one where some of the focus will possibly be targeted.

2.1.4. **Example IMS flows**

In order to better understand how network components work with each other, one could consider a simple scenario where a user registers in the network, with Figure 3[11] representing the flows generated in the IMS core. Although a high-level view on the procedure itself, not intending to give a full description on it, this scenario is useful for creating the proper context around IMS components and their responsibilities. The main concepts to take are that upon receiving a REGISTER request from a user's device, the request is forwarded by the P-CSCF (the first point of contact to the network) to the I-CSCF. This component queries the HSS for an available S-CSCF for user appending, a contact done by calling an HSS exposed interface called Cx and exchanging the Diameter protocol, which is intended for AAA (Authentication, Authorization and Accounting).

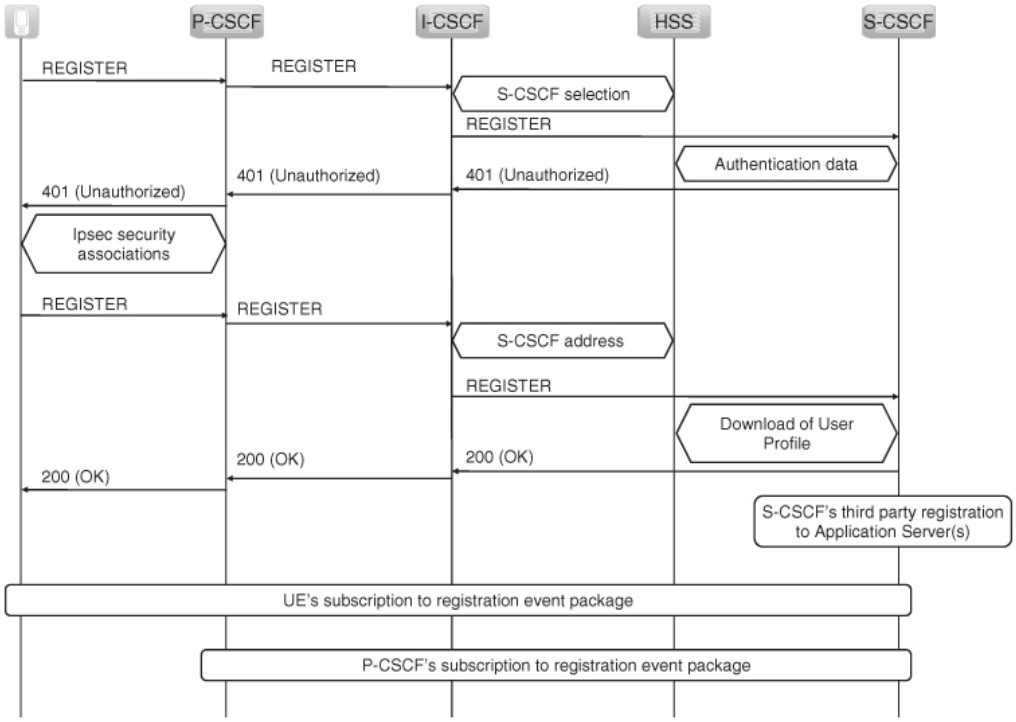

Figure 3 - User registration flow within an IMS core

After this, the given S-CSCF is contacted, this time interacting with the HSS to create an authorization vector which will be sent to the user in a 401 Unauthorized response. The user's device sends a REGISTER request again, this time with the response for the original authorization vector. Essentially, this request goes through the steps above a second time, reaching the given S-CSCF. Now, the HSS is notified of the registration and accesses the user's profile, seeing which application serves it has access to, among other actions. The final response is a 200 OK sent to the user, showing all information regarding said registration.

At this point the user is registered and allocated to an S-CSCF. Posteriorly, and choosing to place a call, a flow similar to the one in Figure 4 will be generated.

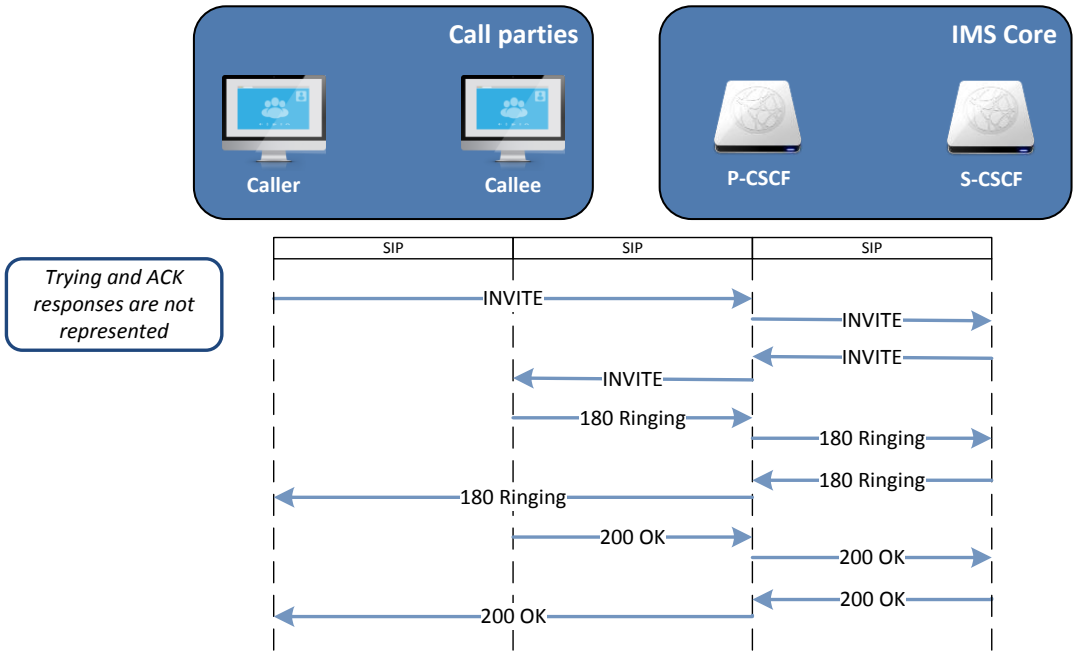

Figure 4 - Call invitation flow within an IMS core

Since both call parties are registered, their location is known and no authorization mechanisms must be made. As such, I-CSCF and HSS are not contacted at this point - considering a scenario where both devices are registered in the same network and allocated to the same S-CSCF. As an invitation comes from the caller's device to the network, it is forwarded by the P-CSCF to the S-CSCF and then to the callee's device. It responds to the invitation with a 180 Ringing, signalling that the device is indeed ringing. Upon user acceptance, it sends a 200 OK to the network, forwarded to the caller and indicating that the call is in progress.

2.1.5. **Voice over LTE**

With IMS only handling signalling and exposing web-oriented services through application servers deployed on the network, proper transportation still as to be ensured, enabling the exchange of SIP request with the network. One recent technological advance in this field, dedicated to the evolution of Internet access and its speed in mobile endpoints, is LTE.

Deployed worldwide as the means of increased capacity and wireless network speed, 3GPP's defined Long-Term Evolution (LTE) is more than just efficiency, but completely redefines a traditional telecommunications network with packet-switching over an IP environment – through a novel core network called Evolved Packet Core (EPC), as depicted in Figure 5.

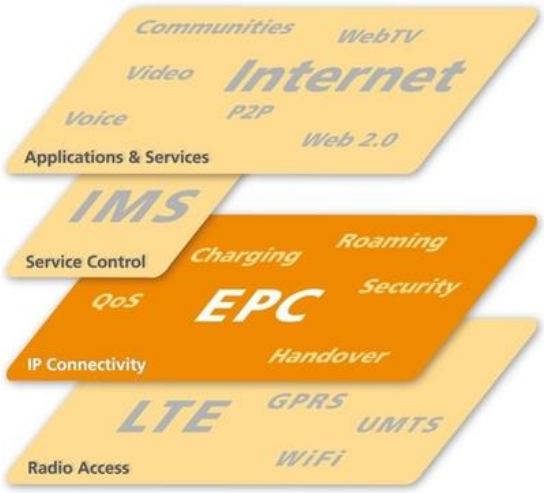

Figure 5- EPC, LTE and IMS relation

From a user stand-point, this will mean that more speed will be available to its endpoint via the cellular network, easing data access. However, more than just a high-speed pipeline for data transfer, LTE is seen as a future possibility for convergence into an all-IP world with VoLTE as a novel effort to exact that purpose. VoLTE, short for Voice over Long-Term Evolution, is an initiative made by GSMA to motivate the standardization of voice and messaging delivery over LTE networks. Although an in-depth analysis on LTE and VoLTE is out of scope, it is important to realize how such advances affect the technologies involved. Analyzing both Figure 5 and Figure 6, one can see how IMS and LTE are related. Essentially the same interactions are done, with a caller's INVITE request reaching the IMS network's P-CSCF and forwarded to the S-CSCF, locating the callee and sending the invitation. However, as depicted in Figure 6, two other components are now featured, certain core elements of the EPC – the S-GW and P-GW (ignoring the PCRF):

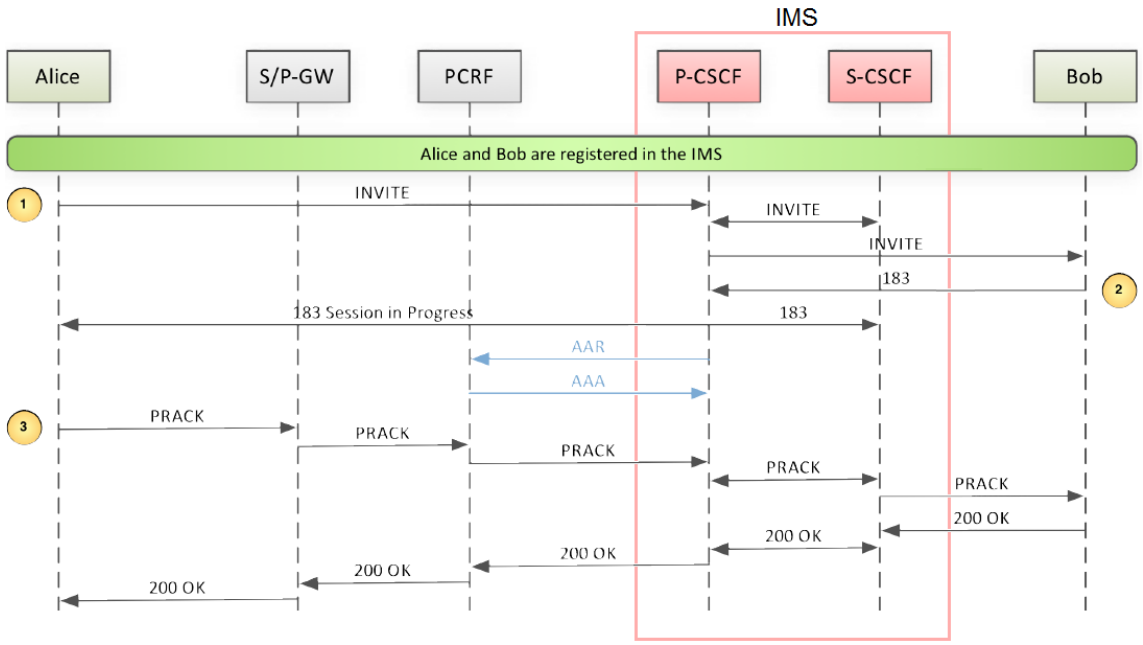

Figure 6 - Call invitation flow within an LTE/IMS network

- S-GW (Serving-Gateway) - responsible for the forwarding and general routing of data packets exchanged by the user's device, it also handles the handover between the LTE realm and other traditional mobile network, 3GPP technologies;
- P-GW or PDN-GW (Packet Data Network Gateway) - enables the exchange between the user's device and the packet network itself, enforcement of security policies and data inspection, by acting as the point of contact for incoming traffic.

Maintaining its purpose of bridging multimedia services and bringing them into an operator's network, LTE is no more than just another transport mechanism (like those seen in topic 2.1.1). Although certain elements such as the EPC and its components have to co-exist, remaining in the middle of the signalling path, these components are dedicated to the interaction between the user's device and its supported mobile network. The network modifications it imposes come to affect the operator's core, but the added signalling in this architecture does not affect IMS in any way[12], since the same SIP and SDP still reaches the core (even though it suffers a few modifications mainly relating to quality of service features[13]). In short, the advances made by 3GPP and GSMA, pushing the speed of cellular networks and augmenting user experience, will not impair this solution of its objectives in the future, being it prepared to handle the core's evolution.

2.2. **IMS core implementations**

There are several IMS cores available in the market, from companies such as Huawei and Oracle's Acme Packet, providing all-in-one or custom-made solutions to fit the clients' needs, solutions that are matured and have been market-tested extensively. The core itself should be "plug-and-play" (i.e. after developing an application server using a core, it should work on a different one due to the standards enforced), so after development one can adapt the solution to any already deployed core. Over the next few sections, known and publicly available IMS cores will be analyzed.

2.2.1. **Clearwater**

Clearwater is a project made available open-source by Metaswitch Networks[14]. Being an "IMS in the cloud" solution, this core (with its architecture featured in Figure 7[15]) contains some components that would be very useful to this dissertation, such as Bono, its WebRTC gateway.

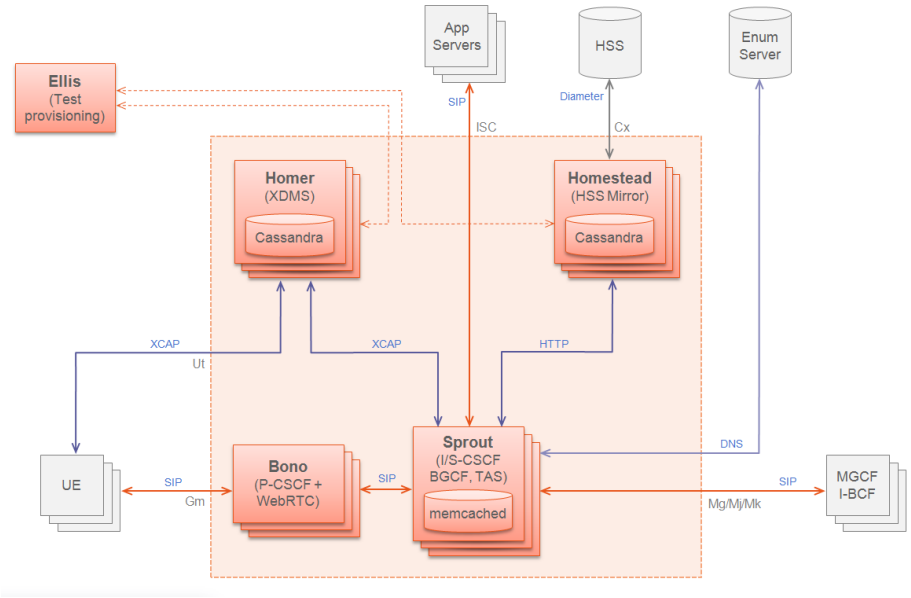

Figure 7 - Clearwater's architecture (with Bono)

From Clearwater's own website[14]: "(Clearwater) natively supports interworking between WebRTC clients and standard SIP-based clients, using SIP over WebSocket signalling" and "you can use Clearwater as the IMS core to support 3GPP standard services such as Voice over LTE (VoLTE) with a suitable TAS and SCC-AS (Service Centralization and Continuity Application Server)".

Even so, this open-source project does not feature certain functionalities that would be of great importance to an operator or to a user - call-transfers are not supported, for example. Its core does not contain an HSS (only an HSS mirror for internal usage) and supports several IMS related interfaces for integration with other components. Although it provides Sprout (a node encapsulating the services of an I-CSCF, S-CSCF and TAS), the telephony services exposed are not suitable for call-transfer for example.

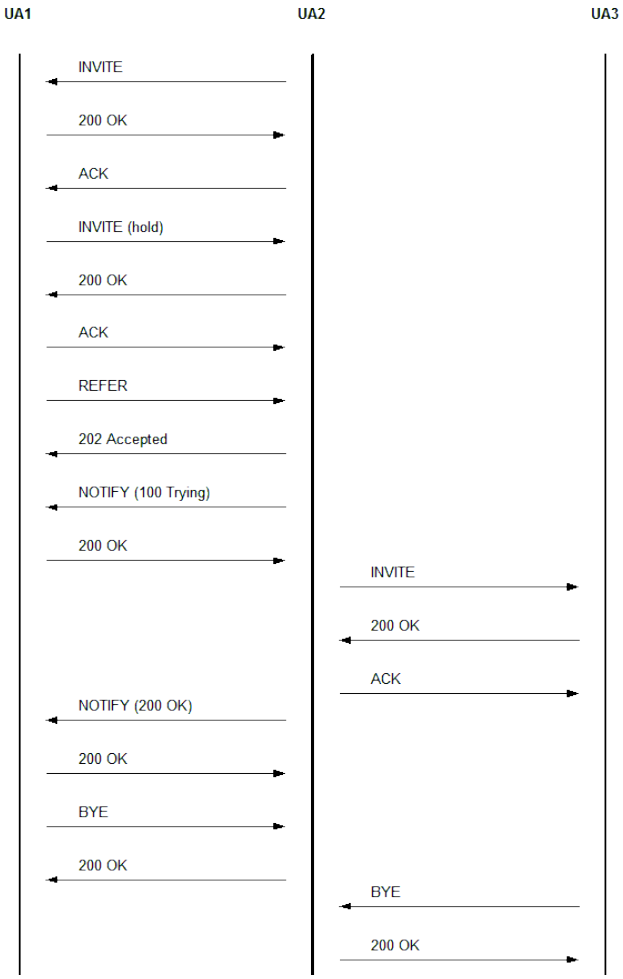

Figure 8 - Blind call transfer example

Figure 8[16] represents a “blind call transfer” where, during an ongoing call between user UA1 and user UA2, user UA2 transfers the call to user UA3, without announcing the transfer beforehand. This operation use of a SIP request called REFER and requires the manipulation of call-legs (for the call transfer), which makes the operation far more complex than simple request forwarding. Since the built-in components and TAS are not capable of handling such scenarios, this mechanism would either have to be implemented or an external, more feature-rich TAS would need to be integrated.

2.2.2. **OSIMS – OpenIMS Core**

An initiative from Fraunhofer FOKUS launched in the end of 2006, the OSIMS (an abbreviation for Open-Source IMS Core) is an open-source product that is being used worldwide as a test-bed for IMS technology testing and application prototyping, providing an IMS core reference implementation[17].

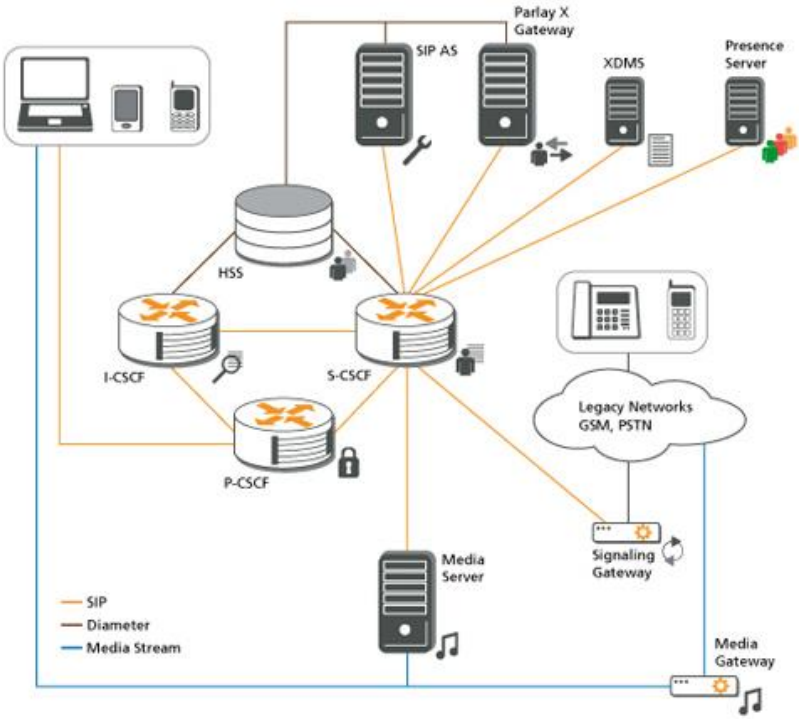

Figure 9 - OpenIMS's architecture

With its creation motivated by the lack of open-source IMS cores at the time, as represented by Figure 9[17], its architecture is composed of all the standard elements discussed above – P-CSCF, S-CSCF, I-CSCF, a HSS called FHoSS (FOKUS Home Subscriber Server), gateways for legacy networks and support for media proxies. Also crucial is the implementation of the ISC (IMS Service Control) interface, allowing the development of applications servers by enabling IMS routing features. However, this core is not fit for production environments, being only suitable as a testbed for development purposes.

OpenIMS has served as the basis for other projects, such as Kamailio IMS, which will be discussed next. Additionally, application development frameworks such as Mobicents provide several examples of communication with this core and tutorials for integrating the FHoSS with Clearwater's core are also available.

2.2.3. **Kamailio**

Kamailio[18] started as a SIP Server. However, over the past few years, with the evolution of the OpenIMS Core, there was a need from the Kamailio project to keep up with the advances made – thus a port was created, featuring IMS functionalities on the Kamailio core[19].

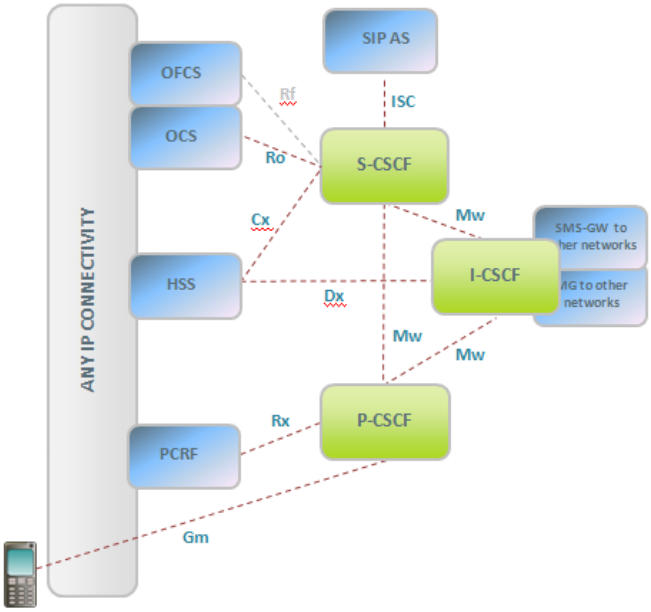

Figure 10 - Kamailio's IMS capabilities

With its architecture depicted in Figure 10, this port has its basis on OpenIMS's components (Proxy, Serving and Infrastructure-CSCFs). However, all other remaining components are not featured. For example, the use of an external HSS is required, components such as SMS gateways or PCRFs are also not present – alike the HSS, external components must be integrated in order to expose their functionalities. Kamailio already implements all the authentication mechanisms needed for HSS exposed interfaces and all the capabilities needed to handle session signalling through the CSCFs. The advantage of this project is the added stability when comparing it to OpenIMS (an unmaintained project, despite community support). Moreover, Kamailio itself is capable of a number of communication features and could be used as an AS in the network to bring other specific features to the network. It exposes a plug-and-play module interface for extension install, featuring a Presence Server and Web Management Interface, among many others. Although not having its own media server, Kamailio works with FreeSWITCH[20], an "open source (…) telephony platform designed to route and interconnect popular communication protocols using audio, video, text (…)", i.e., FreeSWITCH could be used as a Media Server, featuring support for multiple codecs (from G.722 to H.264).

2.2.4. **Core comparison**

Although retaining common core elements and being already available solutions, there are significant differences between the cores. Comparing OpenIMS's architecture and Clearwater's for example: taking a look at both, on Figure 7 and Figure 9, one can see that both have their own advantages - Clearwater has a WebRTC gateway, a strong point towards this dissertation goal, but OpenIMS features a proper HSS. To enable such functionality in Clearwater, one could either purchase available solutions from the company or implement them (Clearwater has the capability to interact with OpenIMS's HSS, for example).

However, since OpenIMS already features this node, development effort on integration of external components is diminished, avoiding possible obstacles by taking advantage of its native support. When studying Kamailio, one sees a similar scenario: it's a SIP Server with Rich Communication Services (RCS) features, gateways for legacy networks and WebSocket support (which could be used for WebRTC handling, analyzed in the next topics). However, it does not include other important elements, meaning that there would still be a need to implement other components in order to correspond to IMS standards.

With this comparison made, the core that will be used for as a testbed will be OpenIMS, because of its features, tutorials, community and influence on other projects.

2.3. **Web oriented RTC technologies**

A telecommunication concept, Real-Time Communications (RTC) refers to the exchange of information in a near-instantly manner. RTC can be seen in various scenarios, from normal telephony calls to Instant Messaging (IM) to videoconference. In the following sections, web oriented frameworks that provide such communication will be described.

2.3.1. **WebRTC**

WebRTC is an open framework for the web that enables RTC in the browser[21], embedding audio/video into it and allowing the communication between two users to be made without third-party applications or plugins, unlike previous VoIP solutions. Moreover, this technology was thought as a way of facilitating the development of RTC solutions over the web, using just one's browser and a simple JavaScript API, depicted in Figure 11, exposing audio/video processing for developers to create their own applications.

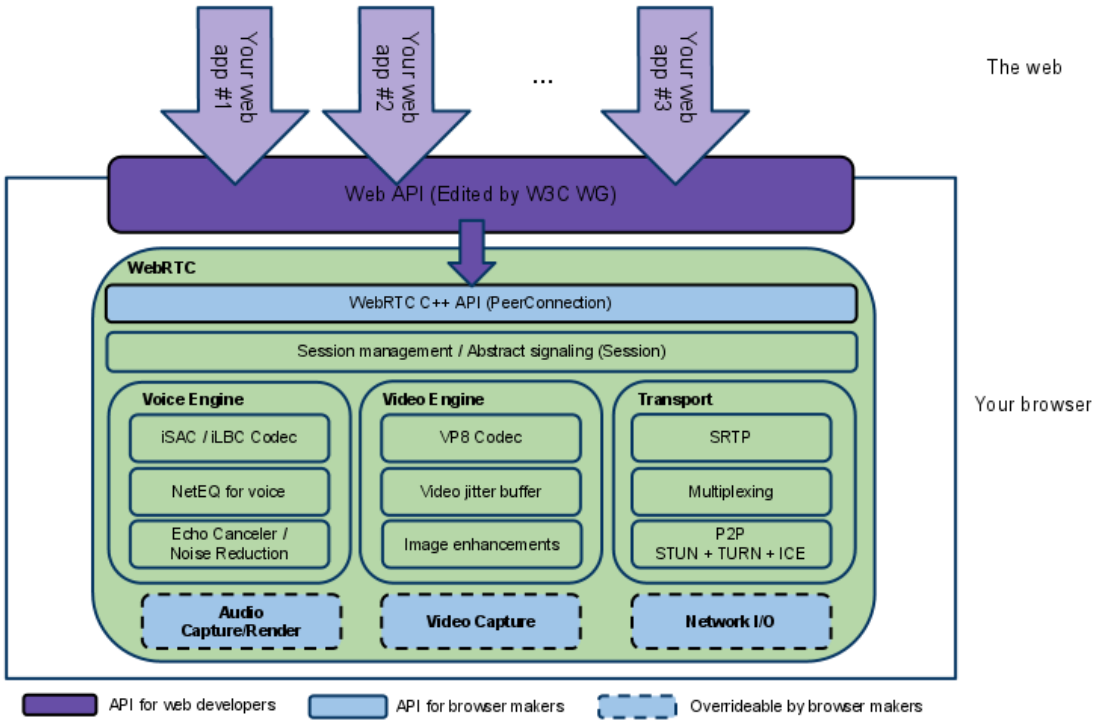

Figure 11 - WebRTC's architecture

With WebRTC, access to computer devices such as microphone, camera and screen is granted, allowing the capture of a person's voice and transmission of the audio during a real-time call, for example (handled by components Audio Capture/Render and Video Capture, in Figure 11). Moreover, these same calls can be established in a peer-to-peer manner, using Javascript-based methods. For audio and video calls, WebRTC's Audio and Video Engine support G.711 and VP8 (among others), although the standard is still open at this time, meaning that the codecs supported may change in the future. The calls themselves are secured using SRTP (Secure Real-time Transport Protocol[22]), an added security layer designed to protect its content by encrypting the RTP exchanged.

More than just audio or video calls, WebRTC features a channel solely aimed at exchanging data, called RTCDataChannel. Essentially, a stream established using a protocol called SCTP (Stream Control Transport Protocol[23]), this channel allows for low-latency data exchange, with configurable aspects such as delivery assurance. As a means of media negotiation and session description, WebRTC also uses SDP as a standard protocol - although this framework still needs a protocol for signalling which the standard deems as undefined. This signalling can be established using custom solutions or, to increase interoperability with IMS networks, using SIP to server signalling. Here, taking advantages of a specification first released in 2014, one could use SIP over WebSockets[24], which "enables two-way real-time communication between clients and servers".

In terms of media exchange, a recurring problem with SIP was the overlook of the Network Address Translation (NAT) process. This process enables the "masquerading" or modification of IP addresses, from non-unique addresses that exist in a local network to unique IP's used for Internet access[25]. Not being able to contact users located behind these translations, NAT traversal techniques were developed, such as ICE or STUN (Interactive Connectivity Establishment[26] and Session Traversal Utilities for NAT[27], respectively), to allow communication – techniques which are supported natively by WebRTC.

In an effort to study what is being done with this emerging technology, how companies have reacted to it and the use-cases implemented, several applications were analyzed, among which:

- Anymeeting - a web application designed for small business users, to power them with video/audio communication, screen sharing and social/chat capabilities[28];
- Talky[29] – a web application focused on anonymous video chat and screen sharing, person-to-person or groups. Using WebRTC, the user doesn't need to download plugins – just create a room (or conversation) and share the link with friends;
- LiveCom[30] – another product, with video conference capabilities, differs from the previous two by enabling the placement of call to PSTN numbers ranging from Argentina to the United Kingdom.

2.3.2. **Other approaches**

There are other approaches available enabling RTC over the web, one of them being a competing standard to WebRTC. Customizable, Ubiquitous Real Time Communication over the Web (CU-RTC-WEB) is a Microsoft competing standard to WebRTC, motivating a discussion over the latter's adopted design decisions. Encountering possible obstacles in interoperability with the usage of SDP, Microsoft has argued that, with CU-RTC-WEB[31], cross-application development will be easier, "proposing a low-level API that is interoperable and flexible" in contrary to WebRTC's API which "is a higher level API that is less interoperable (…) and supports fewer scenarios". Additionally, CU-RTC-WEB supports a wider array of codecs, reinforcing its position on a more flexible solution to web communications[32]. This proposal was submitted to WebRTC's working group in late 2012 and as seen some of the proposed modifications accepted.

However, in terms of market adoption, CU-RTC-WEB as been left aside, with developers focusing on WebRTC, showcased by the growing number of WebRTC based solutions. Another RTC enabler is Flash, a multimedia platform first seen in the late 2000's (then called Macromedia Flash). Flash saw wide acceptance over the years, with cross-platform plugins (Flash Player) made available to all users by Adobe and possible incompatibilities being resolved by the company.

Even with its strong presence, Flash has seen its popularity decrease, especially regarding mobile phones – it's now deprecated and Adobe AIR/HTML5 are being used more commonly. Despite these "set-backs", Flash and IMS interacting is possible, due to the existence of several Flash solutions that can communicate with SIP instances. Among these, one can find Flash2VoIP, a "conventional SIP based video telephone written in Adobe Flash"[33] which can be used to make audio/video calls with support for the H.264 video codec. One can also find Flash2IMS[34], an open-source gateway from Doubango Telecom that can be deployed on an IMS network to provide Flash interoperability.

2.3.3. **Technology comparison**

RTC technologies have been evolving over the past years, with WebRTC and CU-RTC-WEB still standards being defined. Even so, there are presently several differences that can be concluded when comparing one technology to another.

Even though Flash is still a major presence in this field, the rise of other RTC frameworks have seen its popularity decrease - WebRTC is an open-source solution that has been gaining followers since its release[35], although still not as used as Flash[36]. Moreover, being Flash such a matured solution, there are several development tools available to ease application development, whereas in WebRTC's case, there are several frameworks (such as easyRTC[37]) that aim to help development, but are still in very early stages – just like WebRTC itself.

As for WebRTC/CU-RTC-WEB, WebRTC as seen much more adoption than the latter and with Google involved with WebRTC's standardization, Android has been the mobile platform that has supported this framework the most[38]. Market differences aside, WebRTC has a fixed supported codec (VP8[39]) while Microsoft's solution supports a wider number of codecs, which may ease the development of new applications. However, with SDP part of WebRTC's standard, this may ease the connection with SIP networks/equipment.

In summary, due to its growing popularity and the current state of competing technologies, WebRTC is the main objective of this dissertation. It is still being defined (for instance, mandatory media codecs have been in discussion recently) and is very new to the market, which may bring some additional obstacles in terms of implementation and product lifecycle.

2.4. **Web-oriented RTC on IMS**

There are already several components in IMS that play a significant part in allowing web RTC technologies to fully interact with these types of networks. Even so, most of the products encountered are gateways designed to bring SIP to the browser, as a means of media controlling and translation[40]. One of these components is the Session Border Control (SBC). Not seen in the IMS cores analyzed, SBC's provide security and complex network functionalities, able to support mechanisms for transcoding and protocol translation in WebRTC-enabled scenarios (for instance, supporting SIP over WebSockets). Obstacles such as NAT transversal, referred in section 2.3.1, can also be mitigated by an SBC, providing important functionalities for SIP exchanges. Figure 12 represents an architecture that could be implemented for such a scenario, with the WebRTC caller representing a WebRTC endpoint connecting to the SBC.

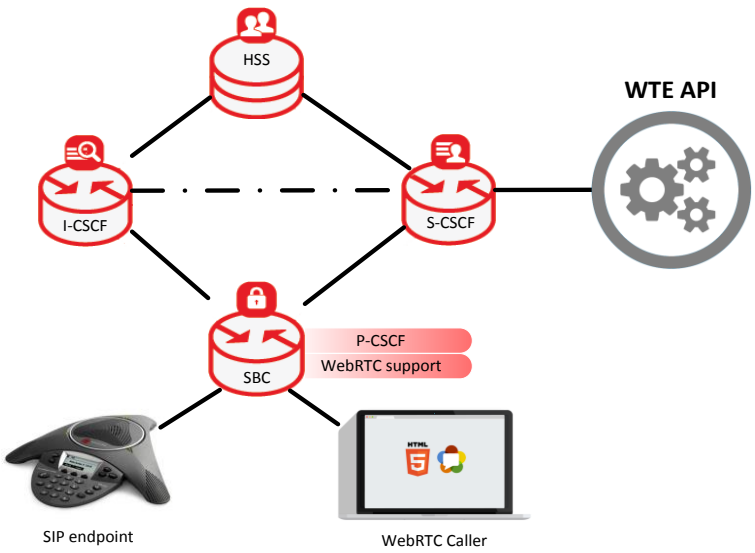

Figure 12 - SBC RTC on IMS solution

SBCs have certain disadvantages that cannot be overlooked: not only is such a solution located between the user and the network, meaning that it will "break the end-to-end signaling path"[41] and media path (adding delay and performance issues to the calls), it also creates a dependency on the network, eliminating the possibility of a "plug-and-play" core, as referred previously on section 2.2. Due to an SBC's complexity and network footprint, there are cases where one would choose not to use one - for instance, for simpler scenarios like click-to-talk functions. In this case, a WebRTC gateway could be used to provide the needed interaction capabilities from the web to the network, with possible transcoding/translation arranged, alongside support for security measures. One may consider Figure 13 as a representation of such architecture.

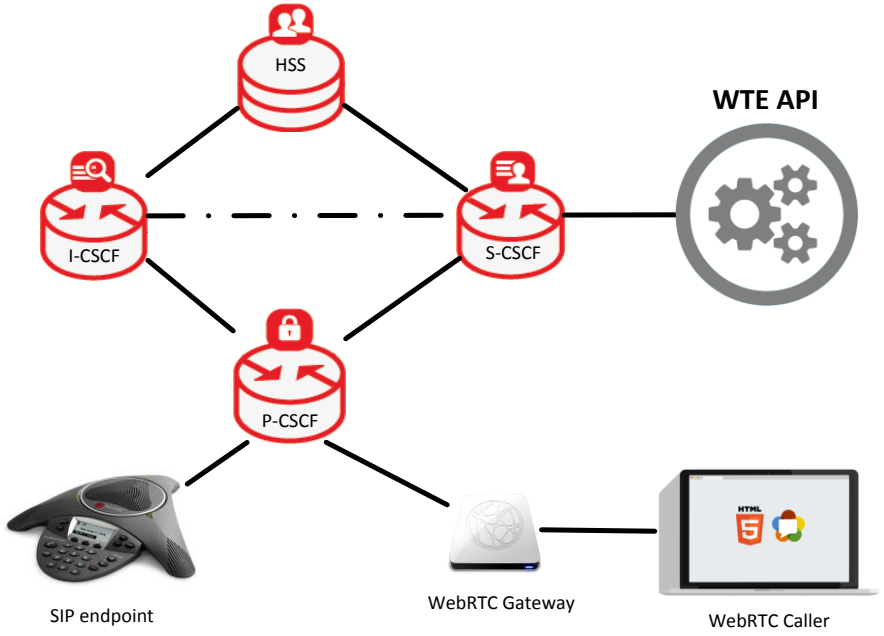

Figure 13 - Gateway architecture for RTC on IMS

With communications passing through the gateway and, after proper treatment, being forwarded to the other endpoints, performance issues can be predicted due to the increased processing demands. On the other hand, this strategy would diminish the footprint on the network that a SBC would bring and provide higher scalability[42] when optimizing the solution.

An alternative would be to create an AS, exposing an API that would provide all the functionality needed to place a call from a WebRTC client to a PSTN phone, being responsible for receiving notifications for new calls and handling them, for example. Figure 14 gives a high-level overview on the exchanges that would be seen in such a scenario when creating a click-to-call use-case.

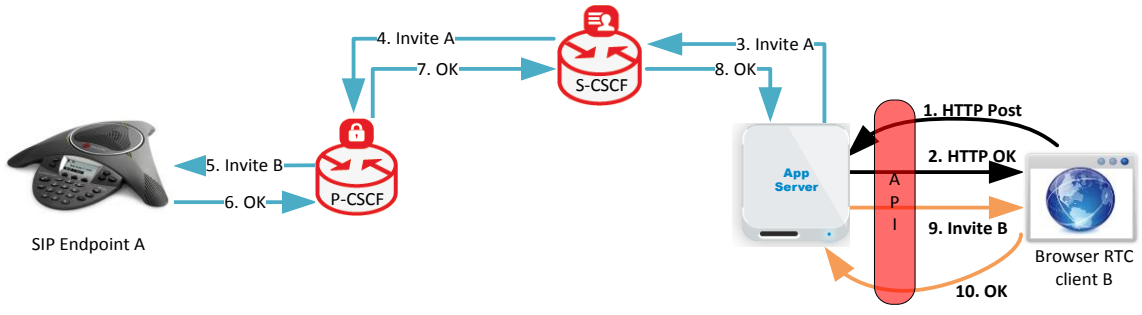

Figure 14 - Click-to-Call flow diagram (initial invite exchange)

In a normal Click-to-Call scenario, a user would click a button on his browser, triggering a call to the desired service. As can be seen in Figure 14, upon the exchange of SIP INVITE messages from the AS to the SIP endpoint, and having the callee acknowledged the INVITE, another INVITE from the AS would be sent to the caller's phone. After that, with both parties accepting the call, a call session would be established. As one can see, with the API located in an AS solution, third-party call controls are possible, with the server maintaining call states. However, as traffic increases due to the service's usage, scalability and performance could become a problem, reducing Quality of Service (QoS) and affecting how other calls are placed.

Implementing an AS requires the analysis of non-trivial technical issues, as it would require the use of a SIP and HTTP container and complex frameworks. A SIP Container is a component responsible for the processing of SIP requests, managing the network's ports for incoming traffic[43]. Upon the arrival of a SIP message, actions are triggered to deal with the request accordingly, answering or redirecting it, for example. The container interacts with SIP servlets, which are managed by a HTTP container, a separate component responsible for servlet life-cycle management. For instance, Mobicents on Tomcat[44] provides an environment for the development of multimedia services, in case an AS side solution was followed in this dissertation. In order to create a telephony API enabling web-oriented RTC calls, Mobicents itself features functionalities that would be of great use to such objective, featuring an HTML5 WebRTC Client and its own Media Server, complementing the network.

An alternative to the previous solution would be Sailfin on Glasshfish[45], which had contributions by Ericsson throughout the project's development. By using Glassfish and augmenting it to provide SIP enabled communication and multimedia services, Sailfin is able to provide multiple tools that would fit this dissertation's needs. However, after acquisition by Oracle, this project was left behind; although still being used, Mobicents has since been used more actively and has gained a strong community as a result.

Either using gateways, SBCs or creating a custom AS, there are pros and cons that developers and operators must be aware of, conditions that affect many network and usage parameters, from call quality, session signalling and security (the methods used to ensure privacy along the call sessions, for example) to drawbacks such as NAT Transversal. There are already several solutions, each falling in one of the categories analyzed in this section, that will be analyzed in the next topic.

2.4.1. **Existing solutions**

Among the solutions researched, those encountered could bring a beneficial insight to the market reaction towards IMS components supporting RTC technologies. Several vendors provide SBC solutions with RTC-enabling functionalities. Among these, Oracle has launched a product called Oracle Communications WebRTC Session Controlled Solution, a combination between Oracle's OCCAS (Oracle Communications Converged Application Server[46]) and Acme's SBC[47], providing the capabilities to enrich WebRTC applications with "carrier-grade signaling, policy and charging"[48].

Focusing on gateways, major vendor's solutions can be found, but open-source alternatives as well, projects such as those from Doubango Telecom, like the flash2IMS gateway and webrtc2sip[49]. The latter is also a gateway that enables WebRTC to SIP network communication, allowing a "web browser to make and receive calls from/to any SIP-legacy network or PSTN"[50]. Using webrtc2sip and sipml5[51], an "HTML5 SIP client" also from Doubango, a user can connect to a SIP network from their browser. With a media stack relying on WebRTC and a SIP/SDP stack implemented with JavaScript, this project comes with several important features, such as screen sharing and call-transfer, all according to 3GPP IMS Standards.

Among the gateway vendors, one can also find WIT Software and their WCAS (WIT Communications Application Server). This is an IMS ready, server platform able to expose converged telephony services across networks and devices, capable of mediating different protocols and integrating different telecom operator sub-systems like billing services (among several others), to provide a more feature-rich solution. The WCAS is able to act as a SIP registrar, for example, enabling the registration of users in the solution. In parallel to the WCAS is another product called WebRTC-to-SIP gateway, enabling (when deployed on the network) the registering of WebRTC clients as endpoints and the translation of all incoming HTTP requests into SIP requests, empowering them with all the signaling and communication needs. Having this in mind, and with the advantages of being an in-house product that has been tested and trialed in this field, with a strong knowledge base available, it will be used to provide the ability for this platform to support WebRTC communication.

2.5. **Telephony APIs**

With the rising adoption of WebRTC[52], the market as seen the launch of several web telephony, WebRTC-supporting solutions, enabling developers to create new applications that would use the companies telecommunication functionalities. In an effort to better understand the market direction (most commonly seen services made available, for example) and competition in this area, a market research has been conducted, with an overview of the competition encountered being featured in this section.

2.5.1. **Twilio**

Twilio offers a REST API to developers so they can empower their applications with communication capabilities. From voice to text/multimedia messages, this service recently started supporting SIP, making it possible to integrate a Twilio powered web application (using WebRTC) with an IMS network.

After experimenting with the service, there are several functionalities (portal and API wise) that stand-out. From dashboard to documentation, this solution's developer portal offers several useful features:

- A developer's dashboard features several analytics graphs (Voice and SMS analytics, showing the usage of both) and quick access to other functionalities with a top-menu. Access to Twilio's numbers bought and billing overview is also available, with information regarding recent payments and usage summary;
- A developer has access to full activity logs, ranging from daily to monthly, featuring the durations of calls, errors that might have occurred and also a "Status" graph, which shows calls that have ended in different scenarios (busy, no answer or failed, for example);
- Triggers can be created to, for example, send an email to the developer whenever a call exceeds a certain amount of time. One can create their own triggers with TwiML - Twilio's Markup Language. When a developer's number receives a call or a text, Twilio is able to execute a previously set script, recording the call, for example.

Drifting from the portal, Twilio features a REST API exposing a rich set of functionalities. From this API, a developer can make SIP calls, arrange conferences, manage already ongoing call sessions, have a caller queued as it waits for its call to be put through or record a call (and have the transcript of the recording made).

In terms of messaging, one can send SMSs (and more recently MMSs), store media objects that have been sent/received or use short codes (numbers that are purposed for high messaging usage) to send several messages. A developer is also able to access his account via the API, verify its numbers and incoming/outgoing calls, receive notifications for events (useful for debugging purposes) and manage a connected application‘s authorization.

2.5.2. **Plivo**

With an API similar to Twilio (a REST API that exposes the inner telephony system) comes Plivo, a San-Francisco based start-up company with an open-source framework for similar telephony objectives - to bring voice and messaging to applications, capable of interacting with an existing telecommunications network.

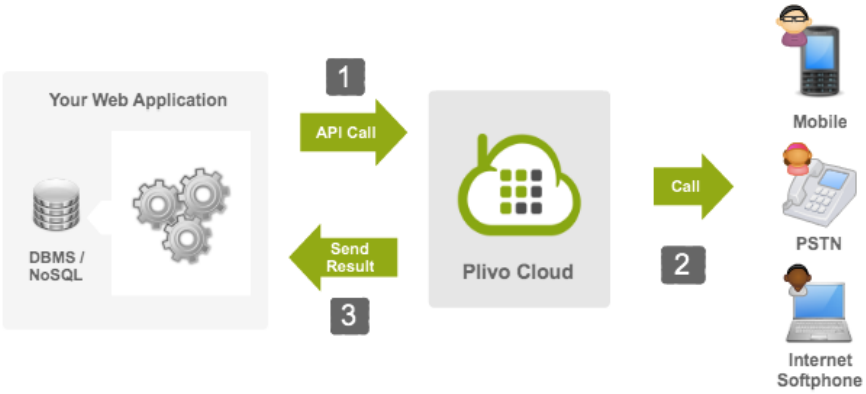

Figure 15 - Plivo's high-level architecture

Featuring voice calling and SMS, support for SIP and public networks, Plivo has some similarities to Twilio, in both API and developer portal. More than just the API, it has on its roadmap the development of a private cloud (i.e., one will be able to deploy a personal Plivo Cloud, as seen in Figure 15[53], for more privacy or redundancy) and Plivo can also connect to a network, although it requires a FreeSWITCH gateway implemented, which can be a problem in some architectures. Additionally, Plivo has recently launched an SDK to enable WebRTC to SIP interaction[54]

In terms of portal functionalities, Plivo also features a dashboard with analytics for voice/SMS (logging of the last 24 hours) and account credentials, having a menu with billing, pricing, logging and numbers bought options for quick access. Also presented in this menu is an option to view documentation, where a developer can find tutorials and SDK/API references. However, the ability to generate notifications over events or to trigger actions when certain scenarios are caught is not available.

On the API side, Plivo implements much of the same functionalities that Twilio does, with other functions included such as DTMF processing (Dual-Tone Multi-Frequency, the ability to send digits over a call), “Play” and “Speak” features which let a developer play sounds or text-to-speech during a call. A developer may also send a recording over a message and is able to manage his endpoints, applications, (sub)accounts, incoming/outgoing carriers or rent new numbers, all through the API. More importantly, Plivo can be viewed as a technological enabler. This company provides an open-source framework[55] for “telephony application prototyping” such as Billing and Voicemail, to be used in conjunction with SIP networks (using the aforementioned FreeSWITCH gateway).

2.5.3. **OneAPI**

This emerging communications market as seen the rise on novelty solutions, such as the ones already analyzed. In an effort to define an API that would provide standardized access to communication services such as call control or billing, the GSMA’s OneAPI[56] initiative has emerged, with the support from several major operators – among which Vodafone, AT&T and Deutsche Telekom. Created in 2008, this initiative has the objective to provide a reference API design for third party call control, payments, location, messaging, device capabilities and data connection description. With its designs residing on an older OMA’s specification known as Parlay X[57], OneAPI offers a subset API based on REST principles, relying on the JSON format for requests and responses, facilitating inclusion in web developments. In its first iteration, GSMA published specifications on textual messaging and multimedia messaging, while in its second one relating to call control. These three are the most relevant API specifications for this dissertation and feature a multitude of methods such as call triggering, add to call, ending of a particular call session and retrieval of its descriptions, among many others in the third party call control part. Focusing on messaging, specifications are available for the sending of text messages and multimedia messages, as well as reception of these in the server and delivery checking.

By using these initiatives and applying them to form the basis of this solution’s API design, one can already mitigate possible problems arising from design choices: these initiatives have already influenced the development of several other products, they have been market-tested and have the influences of major players. Additionally, by using an already known and market-tested standard interface, overall attractiveness of the API is augmented and development is facilitated, mitigating possible shortcomings.

2.5.4. **Summary**

As a reference point for the API's explored, a brief comparison can be seen in Table 1. As one can see, Twilio and Plivo are very similar, differentiating in a few features such as rich messaging and DTMF processing. GSMA's OneAPI does not expose many of the features provided by the others, among which call modifications like call hold or transfer.

	Twilio	Plivo	OneAPI
Call placement	✓	✓	✓
Call modification	✓	✓	✗
Conference calls	✓	✓	✓
Call recording	✓	✓	✗
DTMF processing	✗	✓	✗
Messaging (text)	✓	✓	✓
Messaging (multimedia)	✓	✗	✓
Account management	✓	✓	✗
Application management	✓	✓	✗

Table 1 - Feature comparison

Having these insights on available solutions will be very useful when designing this dissertation's API and portal, realizing which functionalities seem most popular and are already available, how developers will view the solution and defining planned features. One detail that stands-out among the services researched is the ease of use, providing multiple examples and developer focused information on the dashboards (analytics, logging, billing summaries, among others). Twilio, for example, provides extensive documentation and implementation examples for multiple programming languages.

Going from a user-centered overview to a technological overview, all the solutions analyzed employ REST APIs. The reasons behind this decision can be somewhat narrowed down to the role REST assumes in communication, privileging point-to-point communication which greatly adapts to the telephony use-cases here assumed versus a service model better adapted to intermediated services (SOAP). Requests/responses are also "lighter" than those exchanged in SOAP due to the latter's encapsulation, providing easier parsing of the content and reduced bandwidth consumption, for example.

On the topic of request/responses, Plivo and OneAPI send responses using the JSON format, something also possible with Twilio, although using XML by default. Twilio also follows this approach to requests, allowing these to be made using either XML or JSON. The choice between these data formats can be difficult when deploying a new service:

- XML is more verbose when compared to JSON, which increases its size[58] and, consequently, affects bandwidth;
- Depending on the type of information, XML can have an advantage since it supports a wider range of data type than JSON (which only supports text and numbers). However, JSON does have native support for arrays;
- Querying-wise, JSON has an advantage, since the data is stored not in a tree structure, like XML, but in arrays and records[59].

As seen, from the type of service architecture to the data formats used, certain design decisions must be carefully considered, lest performance and scalability be affected as the solution is developed and used. More than what functionalities the API's are able to deliver and the architectural decisions made, the way the API is organized - how it appears to developers and how they are able to use the underlying services - is an important detail that should not be overlooked. For example, if the API is designed in a similar fashion to Twilio's, developers that have used it will adapt better to the solution provided, decreasing its learning curve and development effort.

2.6. **Chapter summary**

Throughout this topic, several concepts were addressed, necessary to a better understanding and proper planning of a solution for this dissertation's objective - to develop and deploy a telephony API that would allow developers to create their own application and empower them with other, already available services such as billing and logging, among others. Moreover, this research is especially useful to understanding how the market reacted to both IMS and RTC technologies, with the first being used as the cornerstone for multimedia services integration in a telecommunications network and the latter being seen as a powerful technology for the future.

After covering and comparing several fundamental aspects of the technologies, components and frameworks related to the context of this dissertation, different conclusions were presented, shaping the next steps of its planning. The next steps will involve requirements definition and system design, from a high-level perspective: the solution's architecture, API design, identification of possible constraints, among many other details.

This overview on the available tools serves not only the purpose of a state-of-the-art overview but additionally as a technical options investigation, providing proper knowledge base for the decisions that will be made during this solution's development. Thinking of a solution where the API would be exposed via an AS, one can see how the OpenIMS core and the Mobicents framework would make an acceptable alternative for test-bed and development. The API itself could be designed following a similar approach to Twilio and applying OneAPI/OMA's definitions, pleasing third party developers that are already familiar with these entities. Moreover, RTC-wise, one can see how Flash was deprecated and is losing popularity, whereas WebRTC is being seen as the go-to technology for RTC. Due to the growing popularity and the possibility of not having to use third-party plugins to place calls, WebRTC could be the web-oriented RTC enabler used in such a solution. Although posing its difficulties, since it still is a working draft, the market has reacted quite favorably to this technology, with new use-cases and applications being implemented every day.

Chapter 3

Requirements for a WebRTC integration on IMS

The research made in the previous chapter was crucial, providing an overview on market status and current trends, such as the exposure of REST APIs by the telephony solutions covered. The solution proposed in this dissertation will now be described in terms of objectives and feature use-cases, how and where the API will be exposed, which features to include, among other details. With this in mind, a high-level view of the solution will now be addressed, analyzing several aspects such as insight into which functionalities will be focused on and possible constraints affecting the implementation.

Following the AS approach for API exposure, Figure 16 represents a proposal of an implementation of such a solution, with each of the components being analyzed over the next sections.

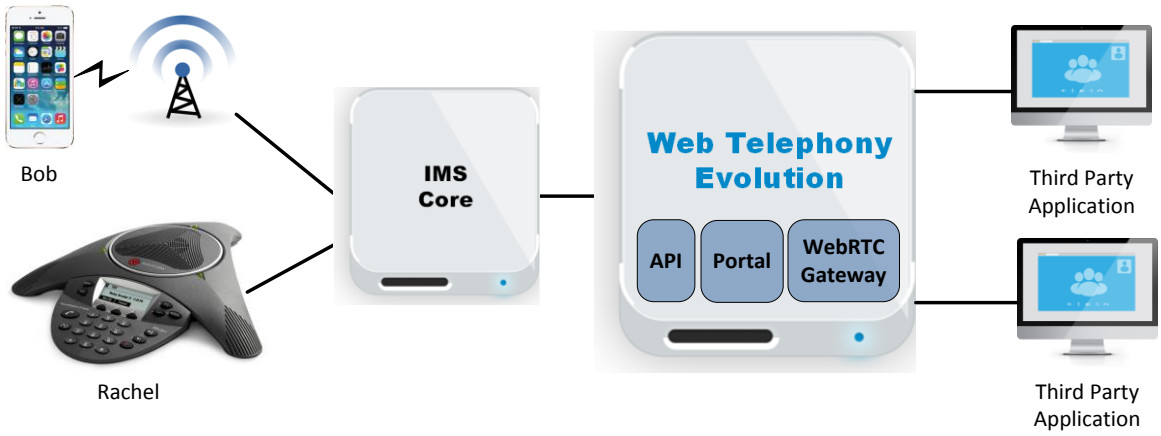

Figure 16 - Logical view of the system

The API will facilitate the creation of applications that allow the placement of calls, conferences or rich messaging to other network endpoints, abstracting internal mechanisms and complexity and keeping the interfaces simple enough to help developers in its usage. Applications will use IMS capabilities (via the API) and will have to be registered in a purpose built platform, the portal.

The portal will enable the control of registered applications by administrators, maintaining policies and managing possible network abuses. In a similar fashion to Twilio, the portal has been designed to leverage the access of different user groups to their dashboards and group functions. In it, applications will be published and featured on an administration managed centralized listing, available to platform users. This listing, the portal's marketplace, will be inspired by mobile marketplaces such as Google Play, enabling the easy discovery of available services by network users.

Support for RTC will come by the form of an integration of WIT Software's WebRTC-to-SIP gateway in the solution. By using this gateway, WebRTC clients such as WIT's Web Communicator (WWC) are able to register as SIP endpoints in the IMS network, enabling them to receive calls from other network registered devices, for example.

To provide all the functionalities needed to create a feature-rich proof-of-concept, the portal and API will offer several services for developers to use in order to enrich their applications. A list of these functionalities was made in annex A) and then prioritized, choosing which features must be present for user trial. Consequently, even though not all features will be available, an overview will be made and will be left planned for future improvements, fomenting a richer solution.

3.1. **Example use-cases**

Before addressing specific requirements or constraints, certain use-cases that the solution must enable will now be presented and analyzed. These will give a more contextualized introduction to the system's high-level functions, the added value it conveys when compared to other already researched solutions and, ultimately, example third-party applications for the solution.

CRM, short for Customer Relationship Management, is a business model designed to help manage costumers and everyday tasks involving costumer service or technical support. This model is commonly implemented in companies by dedicated applications, which help keep track of the appointments made by a sales representative – a small example of its innumerous feature. One could consider a scenario such as the one in Figure 17.

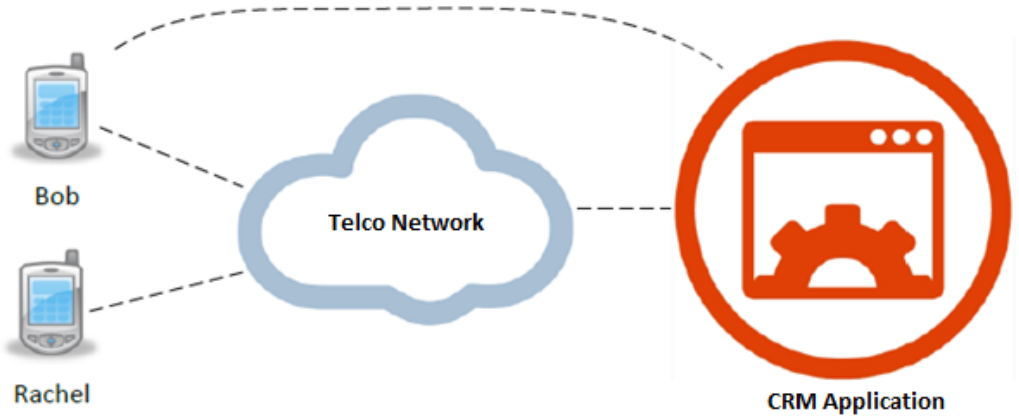

Figure 17 – Calling a client from a CRM application

Bob is a sales representative on his way to a meeting with Rachel. Since Bob wishes to consult the meeting's agenda, in order to get some context beforehand, and call Rachel to confirm its schedule, Bob accesses his company's CRM application, through his smartphone, and checks all the desired information. Since the CRM application uses this dissertation's solution via a previously implemented plugin, Bob is able to trigger a call to Rachel, connecting both devices through an operator's network. After the call, Bob is able to include any notes he may have taken from the conversation on the web application and store them for later use.

In addition to the previous example, first party communications should be also implemented, such as the ability to place a call through the application and answer it in the same webpage. This example can be seen depicted in Figure 18.

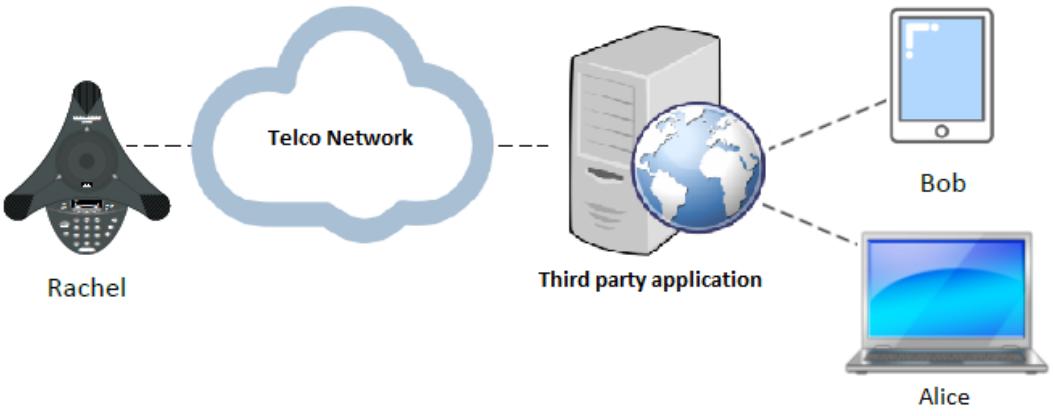

Figure 18 – Adding a friend to an on-going call

As such, a user (Bob), accessing a third party application (that uses this dissertation's solution) on his tablet, may be able to use such application to trigger a call to Rachel, who is using

a SIP phone. Upon answering the call, communication would be enabled between both Bob and Rachel: Rachel would talk on her phone and Bob on his tablet, in the same webpage where he started the call. However, during the call, Bob wishes to add Alice to the conversation, accessing the application using her laptop. Again, with the application integrating this solution, such conference calls would be possible, with these functionalities being abstracted by the API.

3.2. **API features**

The telecommunications API, exposed by the AS, will be structured in accordance to GSMA specifications - voice control[60], text messaging[61] and rich messaging[62] will be modeled after the OneAPI initiative. Making it a REST API means that its stateless characteristics ease resource consumption and impose less processing strain, since handling and maintaining session states is not necessary. On the other hand, this will put additional strain on interpreting all requests and authenticating them. Nonetheless, by ensuring the usage of tokens and notification mechanisms, the API will provide proper session processing/handling, something which was commonly seen in the API's researched. To developers, this abstraction facilitates the development of new applications and reduces the time spent on, otherwise, creating the means to communicate with every single component needed. By simply calling API methods, the developer is able to deliver new functionalities in a fraction of the time.

The following features will be provided, allowing a developer to create rich, telecommunication capable, web applications:

- Authentication - application and user authentication, allowing these groups to use the desired services/resources;
- Third Party Call Control – implemented according to OneAPI guidelines, developers will be able to use call triggering and ending capabilities, along with call modifications such as hold/resume. Mechanisms for call transfering will also be exposed;
- Messaging – users are able to send text and multimedia messages to other registered users, with the methods based on OneAPI specifications;
- Notifications - the forwarding of notifications from the network to the application is available, notifying it of call states (for example, if the call is in progress) and providing all the signaling needed to communicate with the network.

The API will also be able to support rich messaging, i.e., the ability to send multimedia content like vCards (a file format designed for the sharing of contact details from phone numbers to addresses and personal websites) and JPEG images (Joint Photographic Experts Group, a popular image format), in addition to textual messages. As referred, through the features provided by the API, several example use-cases can be idealized like the integration with available content managers or CRM's (Customer Relationship Manager). With this integration, a caller would be able to view its information on a customer upon the triggering of a call. This way, the caller would have access to the background of a costumer, providing better assistance on a service, for example.

The API will focus on third party call controls, with first party being left to SIP endpoints or WebRTC clients. The differences between first-party and third-party calls are significant. Figure 19, an example taken from IETF's RFC 3725[63], will provide the basis for an analysis on what a typical third-party call between (already registered) SIP endpoints would trigger in the network.

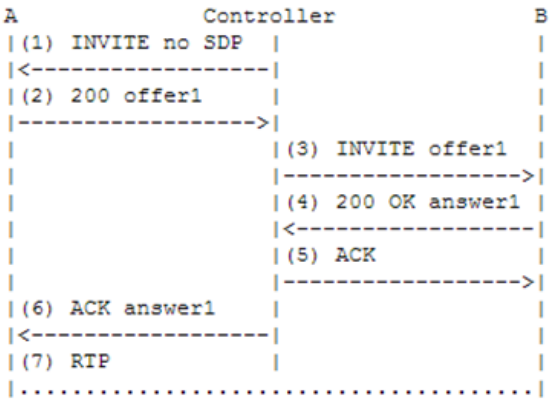

Figure 19 – Third-party call example

One can see how an invite is sent from the AS (represented by "Controller") to endpoint A. After receiving an OK from this party, an invite is sent to endpoint B, something that would not happen in a first party call scenario. If this user accepts the call, an acknowledge response is sent to A and a new call session will be started between both parties, exchanging media. Network-wise, one would see a flow of messages forwarded to the S-CSCF, P-CSCF and then back, with the callee located and informing his decision (if he has accepted or rejected the call, for example). Since the AS plays a central role on call signaling in these third-party calls, it may manipulate the call as necessary – it may, in fact, invite other users to the call, creating a conference. One can consequently say that the AS is acting as a B2BUA (Back-to-Back User Agent), where it remains in the signalling path and calls are actually established between the clients and the server, not directly with each other. This would be done by simply forwarding invite requests to the desired users, establishing the communications exchanges.

3.2.1. **Notification mechanism**

The exchange of notifications between the AS and third party applications is crucial in order to establish the proper means of communication, providing information on incoming calls or call states, for example. To provide such support, different approaches may be taken, from pull to push notifications. This comparison and discussion has been placed as annex E) in this document, with the chosen solutions presented next.

If the client were to announce to the server a URI where he wishes to receive notifications, the server would push the information to the client, instead of receiving multiple connections like planned before. This is called a Webhook-based approach[64], depicted in Figure 20.

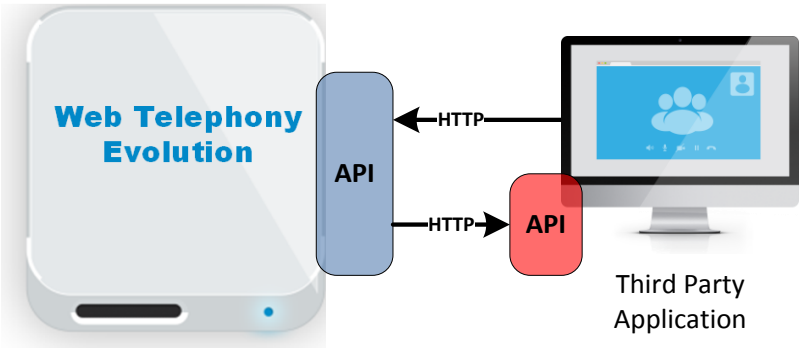

Figure 20 – Webhook notification mechanism

The mechanics behind a Webhook, a recent concept, are straight-forward: upon requesting the creation of a new call, for example, the applications will subscribe to the call notifications by sending an URI to where these events may be pushed by the server. This is merely a REST interface, from the client's side, that receives a specified JSON payload informing of all changes, call statuses or other relevant information.

Nevertheless, this is not a perfect solution. The default Webhook mechanism is ideal when in a scenario of server-to-server communication where IP's are public, but if one was to create a desktop application, for example, and wanted to use this API, the user might suffer of NAT transversal problems if proper care wasn't taken to ensure communication between server and client. As such, a fallback long-polling mechanism will also be implemented in this solution. In long-polling, a client's request is only responded by the server upon receiving new data, timing out after a certain period of time otherwise. This is completely different from the Webhook mechanic explained above, drifting from a "push" to a "pull" scenario.

3.2.2. **API analytics**

Statistics reporting such as API and application usage, as well as analytics per user, may be implemented via Apigee[65] integration. Apigee is a company that provides services for API managing, through an online platform, which acts as a proxy for deployed API's, as depicted in Figure 21.

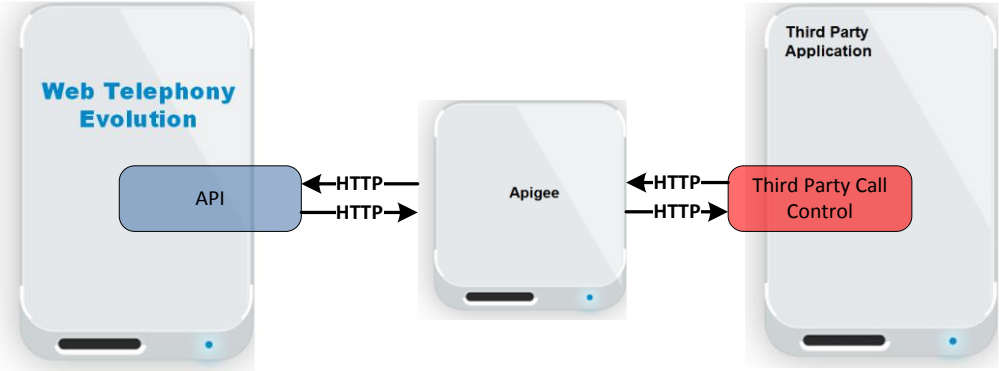

Figure 21 - Apigee as a proxy

With this proxy in place, Apigee is able to provide several features over the incoming requests – features such as analytics of used API resources, filtering through IP, session management allocation of quotas for a given application, among many others. From this product, the solution's developers and administration will also be able to receive analytics on the product itself. Apigee's integration would decrease development effort on features that would, otherwise, have to be implemented from scratch in the solution.

3.2.3. **Integration features**

Deployed on an IMS network, a telephony application server would need to interact with different AS's in order to achieve features such as presence, chat, billing and conference calls. This means that several different servers would be deployed on the IMS network:

- Media server - during announcement playback or conferencing, a media server would be consulted in order to reserve resources and play the desired media or initiate a conference room. This would be done by following a call flow similar to the one shown in RFC 4240[66], where the user would participate in a call with the media server;

- Chat server - a messaging server would be used to provide the messaging needs of the system, with support for rich exchanges, instant messaging and offline message handling (storing the message if the user is offline and delivering it when the user comes online again);
- Billing server - although the AS could generate its own CDR's, a billing server integration might also be required for additional billing features, being consulted via the diameter protocol, for example;
- Presence server - in order to provide presence capabilities and subscription to "friend" lists, a presence server should be deployed on the network.

In this solution, messaging, conference calls and media playback features will be integrated into the AS. Billing, presence and chat features were considered and are intended for future implementation but are of low priority at this stage due to the added complexity and development effort required, having other features been prioritized to provide an interesting proof of concept.

3.3. **Portal features**

A portal is required to provide a back-office similar to those featured by Twilio or Plivo, dealing with different user groups, functionalities and registered applications. These will be available through the marketplace, upon administrative approval and publishing, to all network users. Different user groups will have access to different features:

- Users – these are network registered users, able to use the portal after being provisioned by an operator through a user provisioning API, fully analyzed in annex C). These will be the main consumers of the application marketplace. Features such as access to billing records and activity logs are also considered, placed on their dashboard;
- Developers – having registered for developer accounts, these users will be able to create new applications and submit them for publishing. After submission, developers are able to consult the publishing status on their dashboards. The assigning of new developers to an application is also contemplated;
- Administrators – solely responsible for maintaining the solution (e.g., responding to publishing requests and requests for new developer accounts, disabling applications or banning developers if they infringe terms of service). On their dashboards, administrators will have access to notification-handling sections and will be able to create administrative

applications. These will have access to the user provisioning API and can only be seen when accessing the marketplace via an administrator's dashboard.

With this in mind, the portal must be able to adapt to the different user groups, exposing functionalities only fit to their permissions. For instance, developers must not be able to execute administrative actions and users must not be able to access publishing features or alter application configurations.

Application and user provisioning mechanisms will also be required. Upon the publishing of a new application or registration of a new user, certain provisioning procedures must be engaged, such as proper authorization. Quota definition might also be implemented, meaning that both users and applications would be limited to an X amount of actions per second (call control events, for example). This is a security measure that, defined by the administration, may be implemented in order to prevent network abuse.

Moreover, the usage of an application by a user must be provisioned. Different approaches may be taken in this case, such as manual or auto provisioning. In the first case, a user would have to register in an application, whereas in the later case, an already solution-registered user would be able to use the application immediately. Due to the simplicity this conveys to the user, the auto-provisioning approach will be chosen. However, how the applications and users are registered compose completely different scenarios: whereas in the first case developers or administrators register the applications through the portal, in the latter, a user provisioning API exposed by the portal will be used. This API will be of exclusive use by administrative applications and will allow the provisioning of users within the solution.

3.3.1. **User provisioning API**

User provisioning is a process where the solution will prepare the usage of its services by a new user. In this process, the user will be registered within the solution, will perhaps have its quotas established and will be given authorization to use network related services. This process can also be done in multiple ways - which have been described in annex D).

However, the aim in this is to facilitate user experience and enable the usage of only one set of credentials by the user – the same credentials used to register the device in the network should be the ones used to access the platform. The solution that is proposed is a user provisioning API, exposed by the portal and depicted on Figure 22, which will allow the creation/update and deletion of a network user.

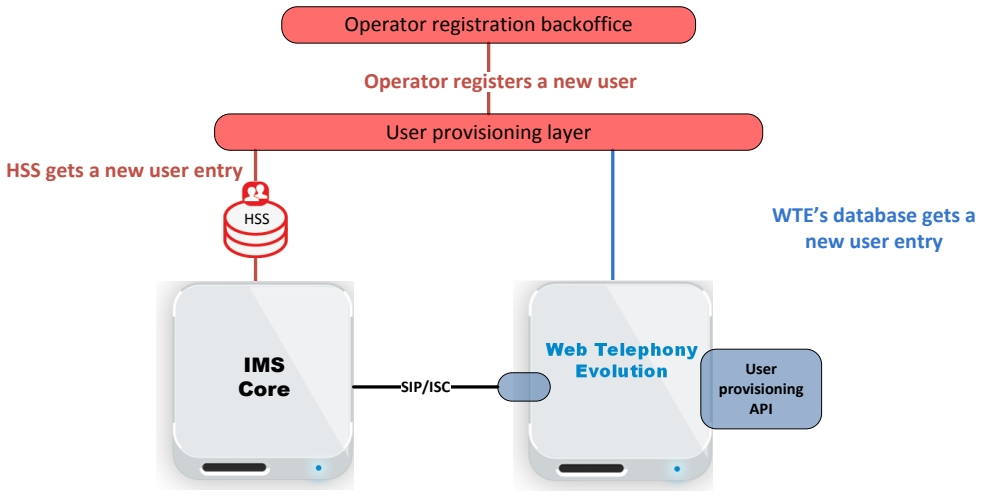

Figure 22 - User provisioning API

This IT abstraction layer will be used exclusively an operator - when registering a new user in the HSS, an operator will also be able to access this API and use it to register a new user in the solution's database. This will allow synchronization between both databases and maintain updated user records. Consequently, when logging in a third party application, the user would use the same credentials used in the network. Unfortunately, this is not without its drawbacks, since the implementation of such a mechanism would mean that user credentials would have to be stored in an external database (with all the security demands that arise), with the operator having to ensure the consistency of the data.

3.4. **WebRTC features**

As referred above, WebRTC support must be enabled in the solution. A WebRTC client will be able to register in the IMS network, with all interactions being translated to SIP requests. Consequently, the client will be able to receive and make calls to other SIP endpoints. To provide the support necessary to establish calls between parties using WebRTC and parties with SIP softphones, a media transcoding service may need to be implemented due to the difference in codecs supported.

Transcoding is basically the direct conversion of one codec format to another, a mechanism that requires a substantial amount of processing power and consequently involves sacrificing either quality or speed. With this mechanism, call placement between parties with otherwise incompatible codec formats is possible. However, not just media transcoding that is necessary, but basic network protocols must be handled as well:

- SIP/SDP are protocols used for signaling and media negotiation and, as such, must be fully supported to provide a robust and functional solution;
- MSRP is the protocol used in messaging - it is of crucial importance when it comes to message exchange, providing the support for instance messaging or multimedia content;
- RTP handling must also be implemented, in order to enable the transfer of multimedia streams between the communicating parties.

All of the above details will be supported via the integration of WIT's WebRTC-to-SIP gateway. Together with WIT's Web Communicator and its SDK, supporting all of the clients' features, first party call control via a WebRTC-enabled endpoint will be possible. Additionally, a multitude of features such as audiovisual calls, file transferring, image sharing, geo-location sharing, DTMF support and call handling (call hold and transferring, for example) are available between WebRTC clients.

3.5. **Network interfaces**

There are two areas of focus when discussing this solution's most relevant interfaces. These include the APIs that the system will expose and the IMS interfaces used, allowing communication between the AS/gateway and the network. Focusing on the IMS interfaces, two will be used, both using SIP as the protocol exchanged and both crucial to the implementation and integration of this solution in this type of IP-based networks:

- The Gm interface that connects user equipments to the P-CSCF server. This interface will be essential to the gateway, enabling users to register as a common device, being able to receive calls and establish communication with other network registered users;
- The ISC interface (IMS Service Control interface) that will connect the application server to the S-CSCF server, notifying the AS's of incoming SIP messages, among other usages. Without the use of this interface, the existence of an AS, able to analyze incoming SIP signalling messages and create new SIP requests on demand, would not be possible.

Drifting from the IMS interfaces to the APIs exposed, two will be exposed to third party usage: the layer for telecommunication capabilities and the user provisioning layer for user registration in the solution. Although not part of the solution itself, a different API will also have to be exposed, in this case from the client side, due to the notification method that has been proposed in section 3.2.1.

3.6. **Non-functional features**

When dealing with a telecommunications solution, like any other system, one must analyze possible constraints that should be contemplated in its implementation. System constraints such as availability, scalability and performance must not be overlooked or an unreliable solution may be created, pushing away users.

3.6.1. **Availability and scalability**

Availability refers to the uptime of the service, demanding the services to be running at all times or suffering the least downtime possible. This constraint is of extreme importance in this context, as downtime in a telecommunications service translates in lost calls and, consequently, lost revenue. Not only available, the service must be able to handle variations in its usage, providing stable and correct responses in a high-load scenario or with an increasing amount of concurrent events - this is referred to as scalability, the ability the service as of handling growing amounts of workload, while maintain its capabilities and stability. Services must be able to accompany the usage required of it, at software and hardware level. At a hardware-level, this could be done by deploying the service in other machines, increasing the amount of machines available to respond to the requests (i.e. horizontal scaling). With more than one server responsible for a single task, load-balancing could be implemented, dividing the load between the nodes. This would also help availability, since if one node was to suffer a crash, the other would take its place.

Software-wise, several implementation nuances must be considered and addressed. Details such as the API being stateless, for example, would mean that less complexity would be demanded from a server design perspective, simplifying it by not having to maintain and clean up past kept states. This would affect needed resources, lowering them since the storing of states and processing of new requests is delegated to other components - a principle called Service Statelessness[67]. As servers discard the responsibility of transitioning states however, processing requirements are increased, since each request contains all the necessary information and the server must interpret all the given information;

The support for concurrent calls to the database is also an issue, necessary to establishing a scalable environment, especially since both API and portal use the database in their actions. Connection pools or cache-like methods will be deployed in order to mitigate the risk of database access becoming a bottleneck. When ensuring security measures, scalability of these servers must also be taken into account, as validating requests, for example, is easier than the rest of the service logic.

3.6.2. **Availability strategy**

Several solutions could be designed to help create a reliable, available and scalable solution. One of these might be to maintain two or more machines responsible for the same function, installing the application server in two different machines, for example – a method called "network redundancy"[68], depicted on Figure 23. This way, in case the AS or WebRTC gateway failed, communication could be forwarded to the other machine, diminishing call loss – a relevant advantage.

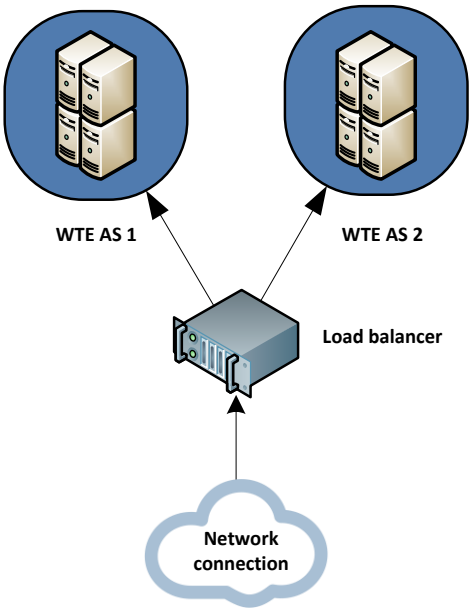

Figure 23 - Network load-balancing

In this strategy, each component will be horizontally replicated, i.e., each node will be executed alongside a backup node, placed under an active/active configuration. With the database being the only common link between both AS and portal, this shall provide three significant additions to the system:

- Workload may be distributed among the available nodes, with the deployment of a load-balancer such as Nginx[69], responsible for forwarding of requests;
- A heartbeat/monitoring mechanism could be implemented, with the load balancer either receiving information from a node stating that it's in fact "alive" or interacting with a node to see its current load, respectively. For instance, if two nodes aren't capable of handling the requests, more may be dynamically deployed to compensate, in a "cold start" fashion. As such, scalability is ensured, with multiple nodes responding to the incoming events;

- Since the nodes would be actively in use, in case of failure of one of them, the system would failover to the other. This way there would always be at least one node capable of handling the network events, resulting in higher availability.

This solution has its drawbacks, trade-offs for a simple yet reliable system. Increased server costs aside, requests being processed by a failing node would be lost, even though incoming requests would be processed normally, with clients being forwarded to the available servers. For instance, ongoing calls which were stuck to the downed server will be lost, with clients obliged to repeat the call – a detail acceptable for a consumer-grade product, but which may prove challenging in an enterprise environment, where the demand for reliability is higher.

3.6.3. **Performance**

System performance will affect how the solution responds to incoming requests, with any bottleneck dictating the amount of calls processed or call setup time. If a system can't handle incoming calls, users won't be able to communicate in peak hours; if call setup times are high, users will become frustrated and confuse it for errors. Availability and scalability measures should be implemented, for scenarios with increased workloads. Certain baselines must be ensured:

- Call setup time must be minimal – from the time a user starts the call to the time he hears the ringing tone on the line, no more than two seconds should pass;
- The system must be able to handle multiple concurrent calls. This value, Calls per Second, must be maximized, with a baseline of 100 CpS being established.

Although testing the solution is required to give more realistic values to these metrics, these established baseline for performance offer insight into how the system is expected to behave.

3.6.4. **Interoperability**

In this solution, SIP and WebRTC powered clients must be able to communicate with each other, increasing interoperability. The support for this RTC technology and for other third-party application must be provided by the solution, empowering these applications with network available resources. More complex scenarios are considered in future iterations of the solution, such as communication between different networks, increasing overall interoperability - providing call support from a caller on the home IMS network to another visiting network.

3.6.5. **Security**

More than just authentication and authorization mechanisms, quotas could be given to both users and applications, limiting the number of requests they are able to make per second. These types of mechanisms are put into place in order to prevent denial of service attacks, where a great number of requests per second are made with the purpose of overloading the system and blocking it from functioning. Additionally, calls between parties should remain private. Using protocols such as SIPS (secure SIP) one can encrypt calls between parties, adding a secure layer to the exchange.

3.7. **Chapter summary**

With the dissertation's goal set on creating a platform exposing a telecommunications API to third party applications and a marketplace showcasing them to network users, one must quickly define all the features intended for implementation. Along this definition, certain questions arise, such as which features will be developed and how the platform will be structured, decisions that come to affect it both in outcome and roadmap.

In this chapter, these aspects are addressed and one can now see how the solution is expected to be shaped and the features it will expose. For instance, the decision to use OneAPI specifications will bring several advantages. These API guidelines have served as the basis of multiple projects and, as such, lower the learning curve for developers that have already used similar solutions. Additionally, these are API guidelines that already been thought for use in different scenarios, lowering the development effort necessary to establish a similar approach. Moreover, with support for RTC via WebRTC integration, the ability to communicate via ones' browser brings great potential, bridging the solution with a novel technology. Nevertheless, key aspects were left out - technical details such as how the features will be implemented were not discussed nor will be covered.

Chapter 4

Solution proposed

With the functionalities that the solution will provide defined, this chapter will focus on discussing the results that were achieved during its development, from the structure of the system to the objectives accomplished – a high-level view on its architecture and what components it will interact with in the network. Obstacles encountered will also be presented and their influence on the solution will be explained, as well as the steps followed to mitigate such drawbacks.

In order to establish a baseline reference design of the solution, Figure 24 is presented, representing the solution's architecture and showing the several components that it encapsulates. Each of these components is responsible for specific features and each will be analyzed in depth in this chapter.

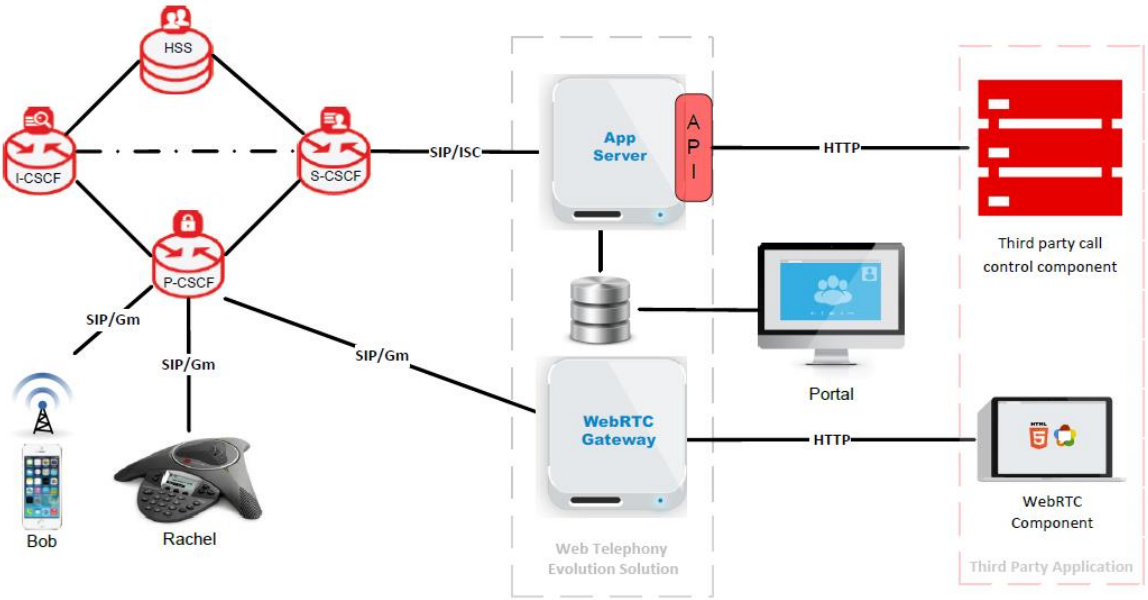

Figure 24 – WTE reference architecture

Firstly, the discussion will fall on how the telephony AS was built and what was used to develop it – from details like the framework in use to the multiple application layers and communication flows. All features implemented in this component are shown, according to the requirements defined in section 3.2.

Secondly, portal results are discussed, showing the technologies used and the application flows implemented. And finally, a view on how WebRTC support was integrated into the solution is presented, with some minor adjustments made to the IMS core in use. Finally, an analysis on which objectives were accomplished and which were not developed at the time is done, comparing the initially planned feature-set to the features implemented. Also, changes to the initial design are shown and deviations from the plan are discussed, showing how the solution was affected during the development phase.

4.1. **Application server**

In order to develop the AS, and due to the time restrictions imposed, features to include where prioritized, aiming to achieve the goal of implementing an interesting proof-of-concept. To ease development effort, a WIT proof-of-concept called Voice as a Service (VaaS) was available to serve as a baseline for the solution, providing many features useful for the dissertation. An introduction to VaaS will now be made, discussing the advantages of extending it and modifying it to fit this dissertation's needs over creating a new platform entirely.

Much alike this solution's vision, VaaS is an application server with several third-party call control features, exposed through a proprietary REST API. It allows the triggering and ending of call sessions, call holding/resume, call transfer, alongside call information and status retrieval, through the use of Webhook mechanics. Additionally, it is designed for media bridging, allowing conference calls and media playback, being able to play a tune for calls on hold. These features would be of great use to this dissertation, since they align perfectly with the objectives and requirements proposed. However, the way VaaS works is significantly different than the way the OneAPI is specified.

Influenced by Twilio's own workflow, VaaS exposes a REST API, responsible for translating HTTP requests to SIP actions, leveraging intended call actions through the use of VaaSML (a TwiML-like markup language defining call session handling). The triggering of a call with VaaS takes the steps seen in Figure 25 and, as one can see, the workflow generated (and the parameters it expects to receive) are significantly different than those expected in the OneAPI. One of the most significant differences falls in the flow expected to exist between client and server – VaaS must receive a VaaSML stating the way it should handle the call, whereas the OneAPI expects only one request to be made to the API, containing all the necessary information to connect the calls.

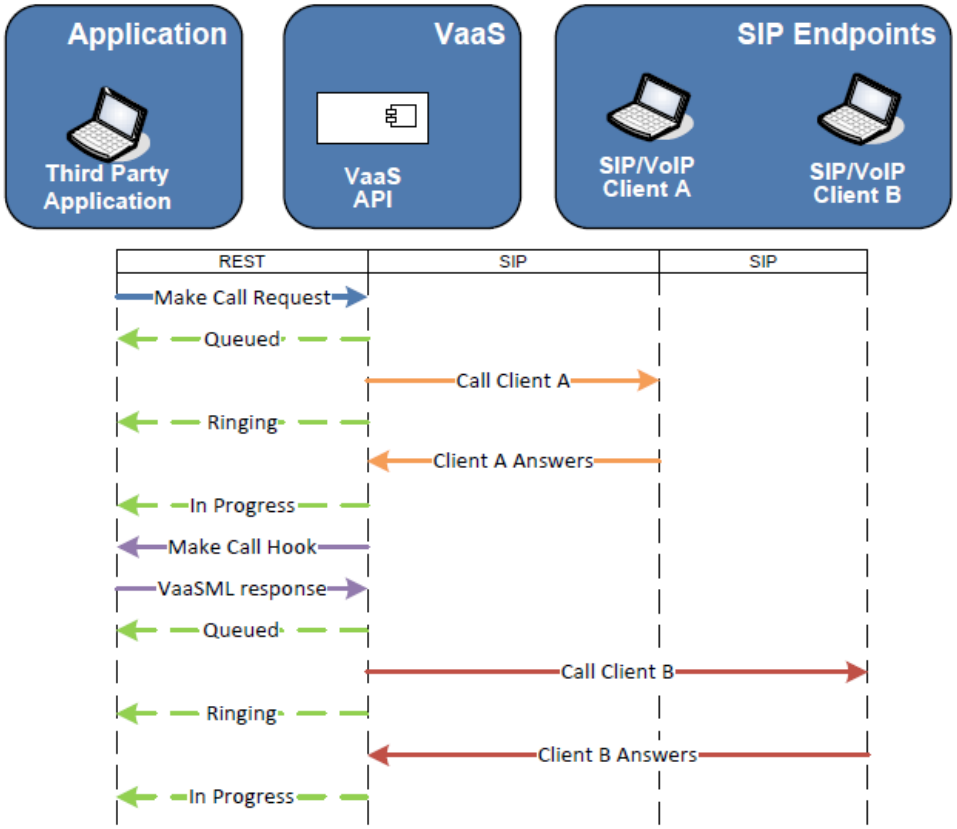

Figure 25 - VaaS' call triggering workflow

From call management to SIP handling, the server already features some useful capabilities. The size and complexity of this solution also reflects the features it exposes. Implemented in Java 1.6, using Sailfin and SIP Servlets (v1.1, abiding by JSR 289 standards[70]), the AS was built in a very modular, layered manner, as one can see in the Figure 26.

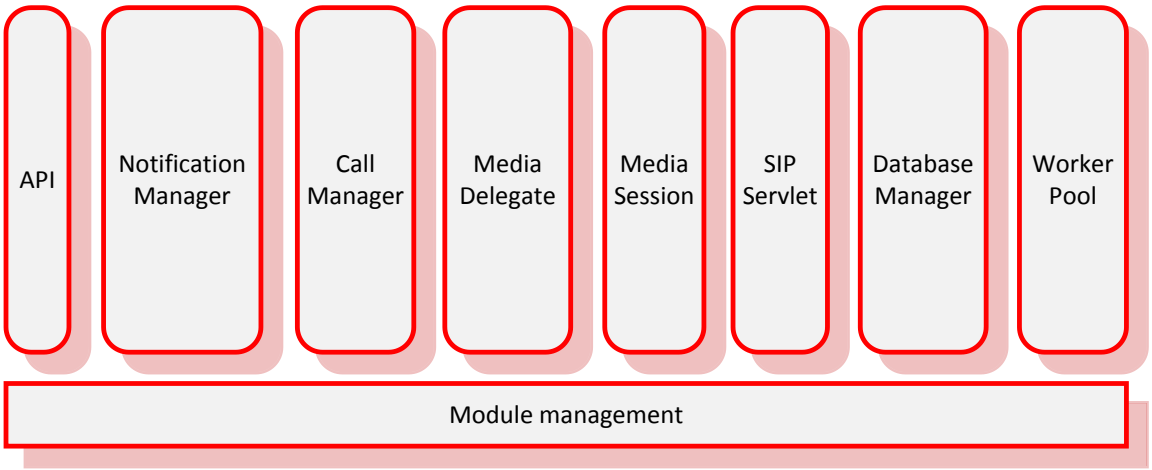

Figure 26 – VaaS' modules

The modules themselves are mostly self-explanatory, with the most relevant being:

- API - a module that refers to the exposed REST API, implemented using the Jersey framework[71], featuring the definition of all resources available;
- Notification Manager – where requests coming from the API get translated into call-specific requests and call notifications are sent to the specified URL's;
- Call Manager – responsible for call session management, which conference to join, invite/re-invite handling and media delegation/session triggering;
- Media Delegation – responsible for maintaining media sessions and SDP analysis;
- Media Session – handling of network actions such as port allocation, with help from the Netty framework[72], for media playback and the media bridging implementation;
- SIP Servlet – where the translated HTTP requests results in SIP exchanges to the network.

Among the most significant advantages and limitations of VaaS, one can emphasize three main details, the first being its call appending as the server allocates calls in media rooms, bridges maintained in the server. Consequently, efficiency and scalability are affected, a crucial detail for any solution designed for multiple, concurrent events. This also creates an obstacle for the implementation of billing features for example – this is commonly done by call leg in an IMS network and, since the calls are anchored in the server, billing mechanisms would grow in complexity. On the other hand, with the session organization that is enforced and call tracking, one gains a certain security layer for handling error events and logging. Additionally, the server only supports the G711 audio codec, with video not being contemplated (other codecs may be supported with some modifications to the media layer).

With VaaS's details in mind, with its advantages and obstacles contemplated, one was able to make a decision on whether to use it as a baseline or create a new solution. Several aspects were taken into account when thinking of a new solution: the development effort necessary, high-level to finer-grain details such as call session management and network administration. Although not a perfect solution, VaaS already had many of these details addressed and dealt with. The discussion fell on whether it would be feasible, in the time-frame expected, to implement such an advanced solution with a stable SIP handling layer. After careful consideration, VaaS was chosen, having identified the areas in which the need for modification existed and how to proceed with such implementations. Taking VaaS and extending it to fit OneAPI's demands involved many modifications to the existing implementation, as well as the creation of two OneAPI related modules. Figure 27 shows the impact it had on the solution, with the round components representing those that are new or were most significantly modified.

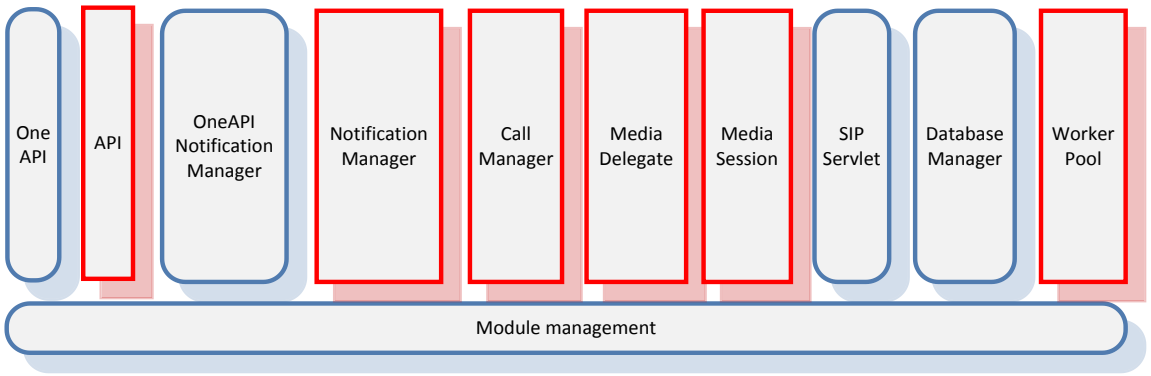

Figure 27 – WTE's AS new and modified modules

When going from VaaS to the WTE's AS (or WTEAS), the first modification to make was the changing of the Sailfin to Mobicents framework. As already stated, this framework has several advantages, development and community wise. This change meant several libraries had to be replaced. After replacing and executing regression tests, VaaS was deployed on an Apache Tomcat 1.6 server, working correctly.

The decision to replace the Java version in use was also made, going from 1.6 to 1.7, a more current version. However, this modification brought unexpected results in the media playback functionality used in call holding, which heavily relied on methods that were deprecated between versions. Consequently, this change was reverted, since modifying the media layer would take significant development effort, which might compromise the implementation of other more important features.

With these changes made, development started on the OneAPI modules, analyzing the current implementation and deciding how the current features may be used.

4.1.1. **OneAPI layer**

Similar to the existing API module, this module was created to reflect OneAPI's resources and methods. Figure 28 represents its multitude of new methods, from call triggering to notification retrieval and user authentication.

The documentation for this layer can be seen in annex B), with all methods exposed and design decisions described. Although the methods' functionality may be inferred by their name and expected OneAPI features, there are 3 methods that are outside GSMA's specification:

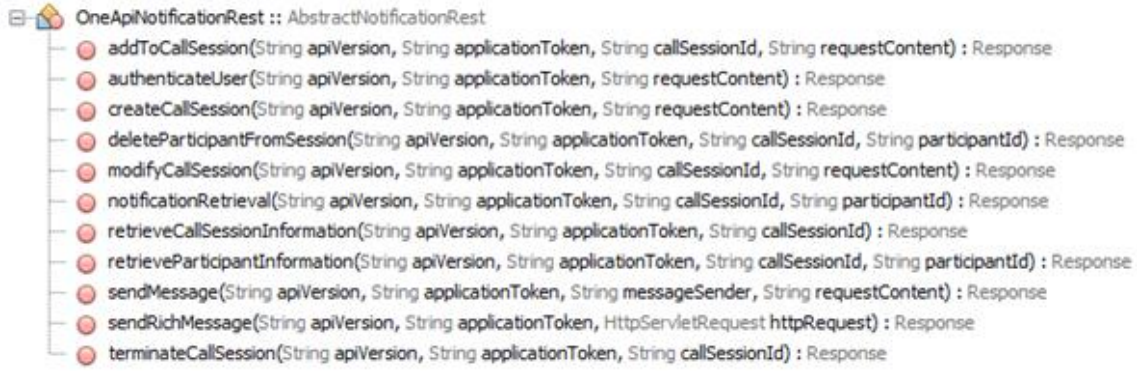

Figure 28 - OneAPI layer methods

- authenticateUser – this method will be used to verify if the given user credentials are correct, enabling user validation to third party applications;
- modifyCallSession – used when applications request the API to place a call on hold or resume it, which was not contemplated in the OneAPI;
- notificationRetrieval – when notifications reach the AS, they are stored in a message queue for a limited amount of time. The API contemplates a resource where applications may request the notifications received for a given call session, using this method. This creates the alternative notification retrieval mechanism, the long-polling method, a fallback for the default Webhook one.

Additional methods and resources may be easily created, following the same patterns and structures used for this implementation. Here, future iterations may take place and other OneAPI features such as billing or location services may be integrated, enhancing the solution.

4.1.2. **OneAPI notification manager**

Whereas in VaaS the solution dealt with call legs and conference rooms, in the OneAPI specification these are mapped to participants and call sessions, respectively. Although these concepts may be similar in purpose (call legs/participants and conference rooms/call sessions), this means that certain changes must be made to the original logic, with new features requiring support from this layer. As such, a new module was created to reflect these changes and demands, with the new modules visible in Figure 29.

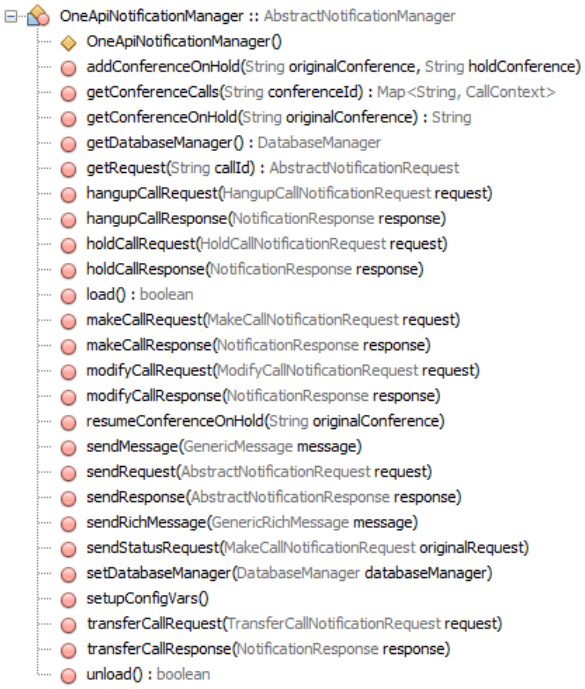

Figure 29 - OneAPI Notification Manager methods

With this decision and with the new OneAPI model alongside the existing VaaSML one, both APIs may work concurrently, without compromise to performance or solution stability. As referred above, although the original logic may have suffered some modifications to fit the present needs, it was mostly adapted and re-used. Across the other modules, certain changes were made, supporting the new features. The media layer, worker pool and original API modules were left untouched – in fact, as referred before, the server is able to expose both APIs concurrently, serving VaaSML and OneAPI clients if need be. Nonetheless, changes to the interfaces specifying the call and notification manager modules, necessary to enable the new features, meant that minor modifications had to be made to these layers. For instance, in the module management layer, the only necessary modification was the registration of the new modules, which had to be loaded alongside the existing ones.

4.1.3. **Database Manager**

VaaS did not interact with any database originally. However, a database module was implemented using a WIT library that serves as a helper for database connections, with connection keep-alive and caching methods. This module was developed as an example for future iterations of the solution, easing development effort. In this AS, the server needs to properly authenticate users and applications, through their tokens. Due to this demand, new methods were made to interact with the portals database and extract the necessary information, with Figure 30 representing them.

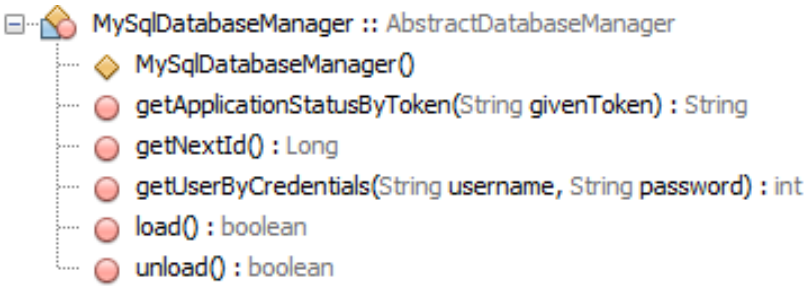

Figure 30 - Database manager methods

Among these, one can find methods "getApplicationStatusByToken" and "getUserByCredentials" (used to authorize applications and authenticate users, respectively) and methods "load" and "unload" **(**used for module management, loading and unloading of modules). The first two methods are used to support the implemented authorization mechanisms:

- The "getApplicationStatusByToken" method is used upon the receiving of a new API request, checking the application status and authorizing or forbidding the request to be processed;
- Method "getUserByCredentials" is the one called when the API's "authenticate user" resource receives a request, comparing the given user credentials to those stored in the database.

4.1.4. SIP Servlet

New features such as rich-messaging made it necessary for the SIP servlet module to handle new exchanges and request types. As such, modifications were made to this layer to reflect such demands:

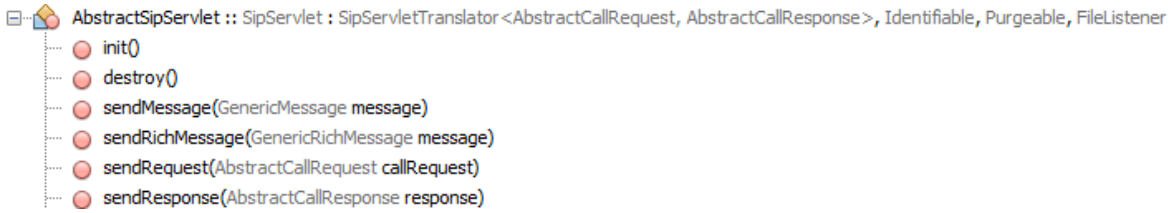

Figure 31 - SIP Servlet methods

Not all methods in this class are represented in Figure 31. Here, the only methods done specifically for this solution are “sendMessage” and “sendRichMessage”, used for sending textual and multimedia messages, respectively, through a SIP MESSAGE request. For instance, a rich messaging request coming to the API goes through the notifications manager, call manager and SIP servlet handler. Here, the correct content-type is established and the message is sent to the IMS core, being delivered to its destination afterwards.

In order to better understand the stance the solution takes in the network and the way the different actions will shape system interaction, certain use-cases will now be shown – use-cases that mirror basic functionalities that are the foundation to user communication.

4.1.5. **Authentication flow**

Authentication mechanisms are crucial to properly securing the solution from exploits and to uniquely identify all platform users and applications. For both users and applications, the authentication procedure follows a flow similar to the represented in Figure 32.

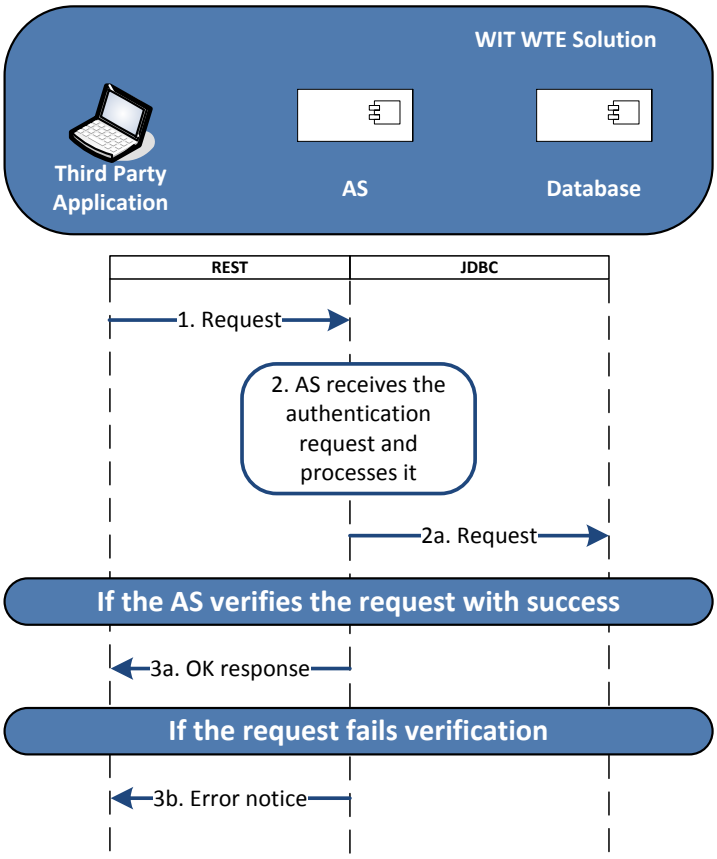

Figure 32 - User/Application authentication flow

There are subtle differences, however, that define both scenarios. As users log-in through third party applications, these are responsible for requesting the AS to verify the given credentials. The AS, in turn, returns a simple true or false, verifying that the credentials match those stored in the database. On the other hand, now focusing on the applications themselves, their tokens are sent in the requests made to the API, leaving the AS to authorize and execute the requests, if the tokens sent have the right privileges (or even match the existing database records).

4.1.6. **Call transfer flow**

In terms of call transferring, this is made in a blind manner where, upon starting the call transfer, the current session will always be terminated even if the transfer destination is busy or unavailable. Unlike a consultative call transfer, this one is done unannounced. A flow similar to Figure 33 is triggered when transferring an on-going call between client A and B to a client C – media renegotiation may be necessary.

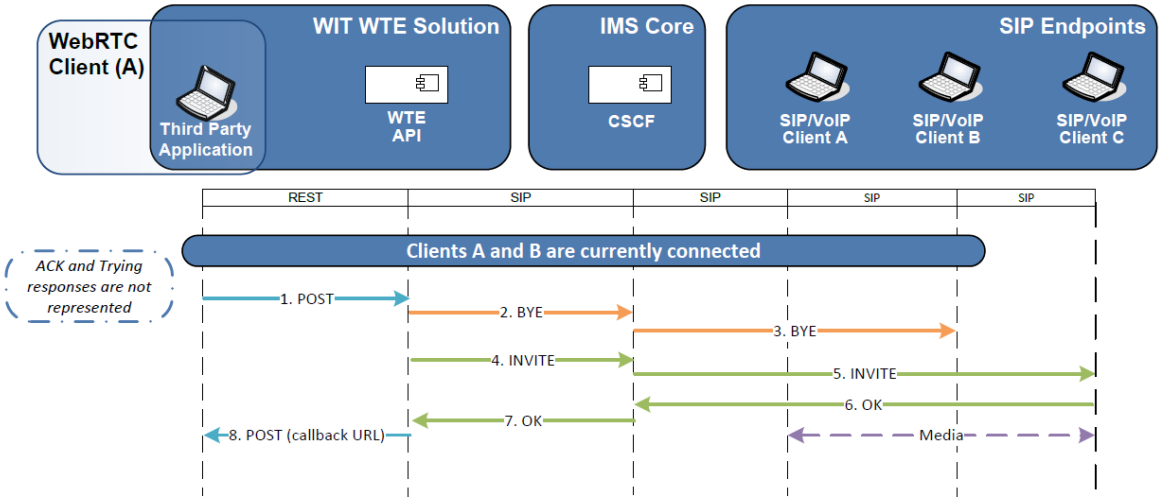

Figure 33 - Call transfer flow

Along the flow, as the calls are set-up or broken down, the solution notifies the client of the actions, creating HTTP POST to the URL that has been defined for callback or saving the messages for long-polling retrieval.

4.1.7. **Call hold/resume flow**

In this scenario, and abiding by to RFC 5359[73], during an on-going call session, the call will be put on hold by renegotiating the call characteristics using SDP, as seen in Figure 34.

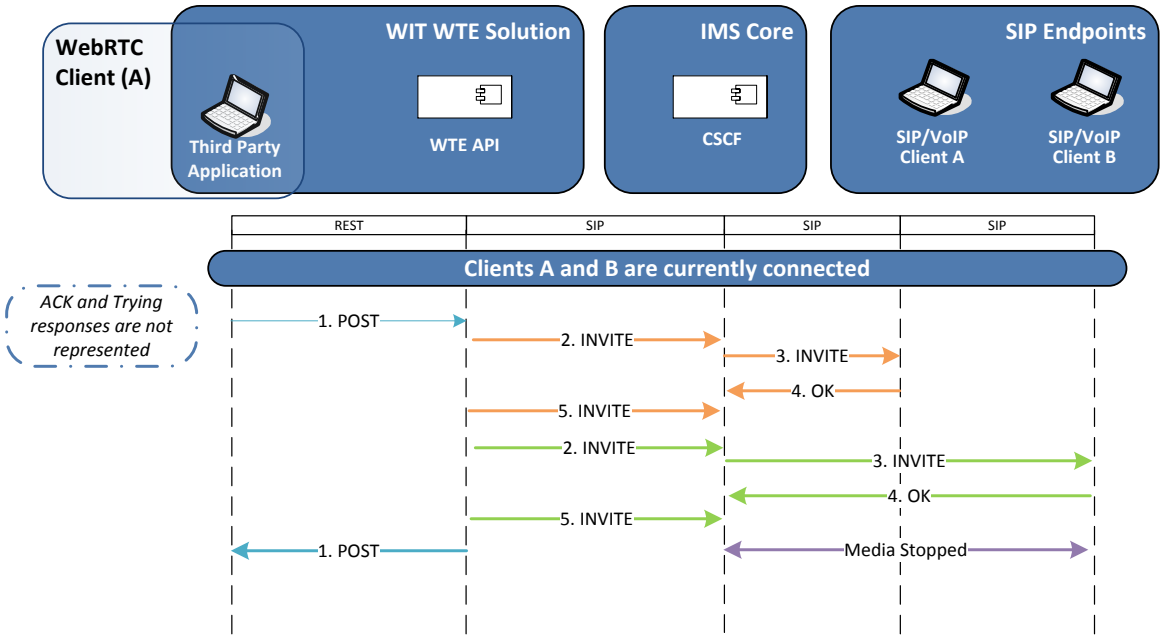

Figure 34 - Call Hold flow

1. The third party application sends an HTTP POST to the solution, wishing to put the on-going call on hold;
2. This results in the sending of a SIP request, by the solution, to both parties, renegotiating the call for both to only receive media ("recvonly", enabling the endpoints to receive call hold music);
3. After this successful transaction, the solution notifies the application of the event occurrence.

To resume the call, another renegotiation procedure takes place, but enabling two-way media streaming once again[73].

4.1.8. **Messaging flow**

Since chat relies heavily on presence, chat functionalities have been disregarded at this stage, having been left to a future iteration of the solution. Nevertheless, just like a call flow, a chat session flow follows a very similar path. These sessions intended for messaging, a mode called Session-Based Messaging[74], are just alike call sessions, but use different media exchanges – instead of RTP, MSRP (Media Session Relay Protocol) would be exchanged. In case of a SIP MESSAGE request (or Pager-mode messaging[75]), the flow would suffer some differences.

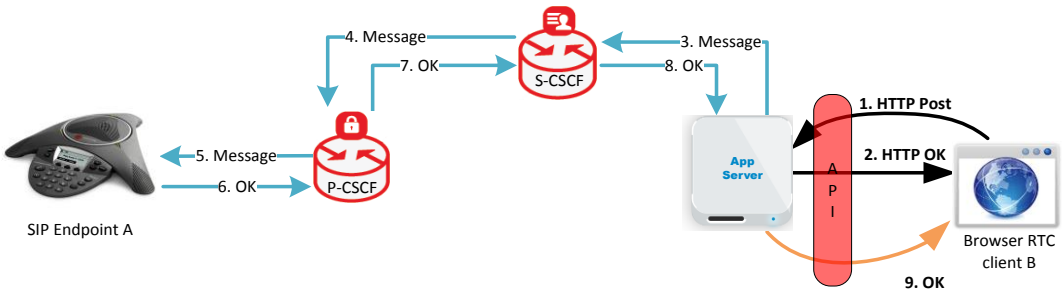

Figure 35 – Pager-mode message flow example

In this case, a chat session wouldn't be created: only the message sent by the sender would be delivered - Figure 35. A web client would send his message via the API, being forwarded accordingly until reaching its destiny. Upon successful arrival, the endpoint would respond with a 200 OK, forwarded to the sender, notifying of its delivery. This is the approach taken in this solution to exchange both textual and rich messages.

4.1.9. **Notification flow**

The default notification mechanism is based on a Webhook[64] approach, where the client specifies an URL where he wishes to receive information For instance, a messaging application triggers the flow in Figure 36 upon sending a new message.

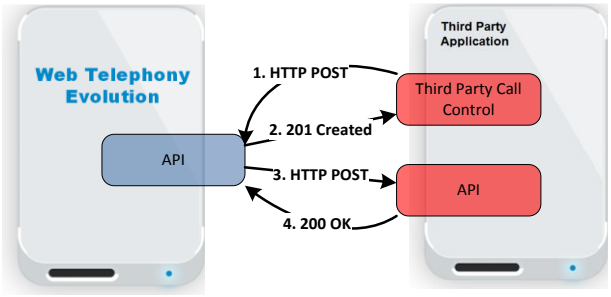

Figure 36 - Webhook notification flow

An HTTP POST, from the third party application to the API, is sent (1) specifying the message body, address, among others, and the notification URL. The solution then responds with an HTTP 201 (2), stating that it has created the message. Upon sending the message, the solution responds with an HTTP POST to the given URL with information stating that the message has indeed been sent to the desired party (3) and the application responds with a 200 Ok, for example, stating that the message has been received. The user can also choose to use the long-polling notification handler instead, retrieving queue messages and generating the flow in Figure 37.

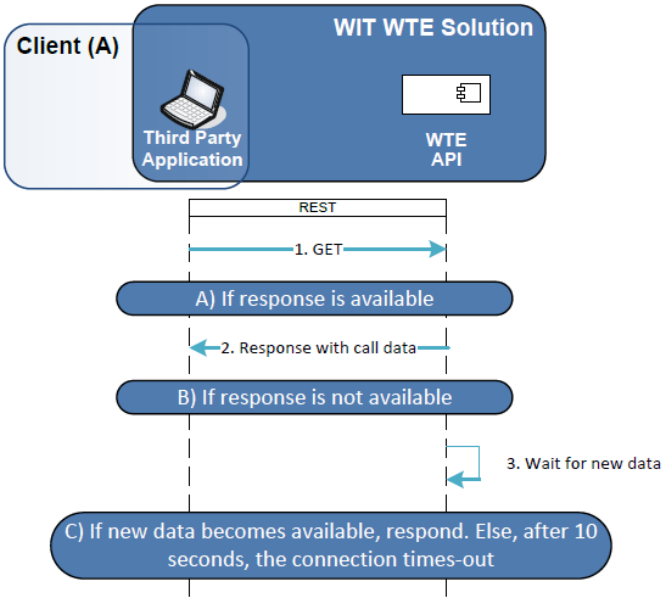

Figure 37 - Long-polling notification flow

4.2. **Portal**

Made using Java and the Struts 2 framework for web application development, the Portal, depicted on Figure 38, was implemented from scratch to fit the needs of this dissertation for a backoffice supporting user, developer and application management, through administrative control. Several features are exposed, among the user groups supported. Developers are able to register in the platform, publishing applications and consulting their statuses. Both requests (developer and application registration) must be approved posteriorly by administration, grating authorization to use the solution. Administrators may also register applications (referred to as "administrative applications", which are not seen by developers or users in the marketplace) – a concept created for those applications that wish to use the user provisioning API.

Figure 38 - Portal's screenshot

Similarly to VaaS' modules, the portal is composed of 4 main ones (depicted on Figure 39) that enable all the features discussed, each responsible for its own set of functionalities:

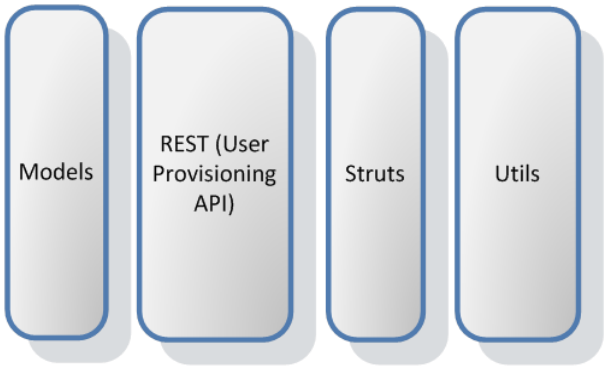

Figure 39 - WTE Portal's components

- Models – this is a component where all the classes used are established. Here one can see what a user or application object is composed of, among other details;
- REST – the component where the User Provisioning API logic can be seen, exposing all the methods available and the logic through which the received data goes;
- Struts – all the logic that forms the web application itself, such as button actions or page routing, is established in this package;
- Utils – certain global functions are needed by the application, such as login validation and database access. These functions can be consulted in this component.

Although almost all of the portal's features have been successfully implemented, there are still some functionalities missing, considered of lower priority – mostly user-centered features. For instance, user accounts have been left simplified, with no place for personal information such as names or dates of birth - as a result, mechanisms for account detail edition are also missing. Developer opt-out (enabling a developer to delete the account) and work groups, where many developers share responsibility for the same application, have not been implemented and neither as activity logging. Thus, dashboards are kept very simple, with room for improvement.

Some features that were not in the initial plan were implemented, such as the administrative applications and application registration by an administrator. This was done due to a growing need to separate applications using the above API and applications from third party developers, which should not use this API in their actions. The API itself is a crucial part of the solution, enabling an operator to register a new user in the solution.

4.2.1. **User Provisioning API**

The user provisioning API exposes CRUD operations regarding network users, enabling their provisioning in order to use the services provided by the solution. Similarly to the OneAPI layer referred above, the documentation for this API can be seen in annex C). An example on user creation can be seen in Snippet 2, where a user may be created by sending a request to the API with the depicted parameters, in either JSON or XML.

```
{ “subscription”: {
“applicationToken”: ”APP_TOKEN”
"user": {
      "impu": "ana.cruz@ua.pt",
      "msisdn": "1236547890",
      "firstName": "Ana",
      "lastName": "Cruz",
      "password": "testing" } } }
```

Snippet 2 - Example user creation body

In this case, the “impu” parameter refers to the users’ email and the “msisdn” to its token. The MSISDN (Mobile Subscriber Integrated Services Digital Network-Number) is a random number, operator generated, which identifies a subscription in a network and, since this API is directed towards operators, was considered for replacement, predicting posterior compatibility issues.

The API was kept simplified, much alike the portal, with the focus being on delivering voice and messaging features, with less attention on portal functionalities. As such, more attention was given to its workflows (developer and application management, analyzed over the next sections) and basic storage needs, than complex user interactions.

4.2.2. **Developer registration**

The developer registration procedure is separated into two different flows, depicted in Figure 40 and Figure 41. First, a developer lands on the page and request an account in the platform.

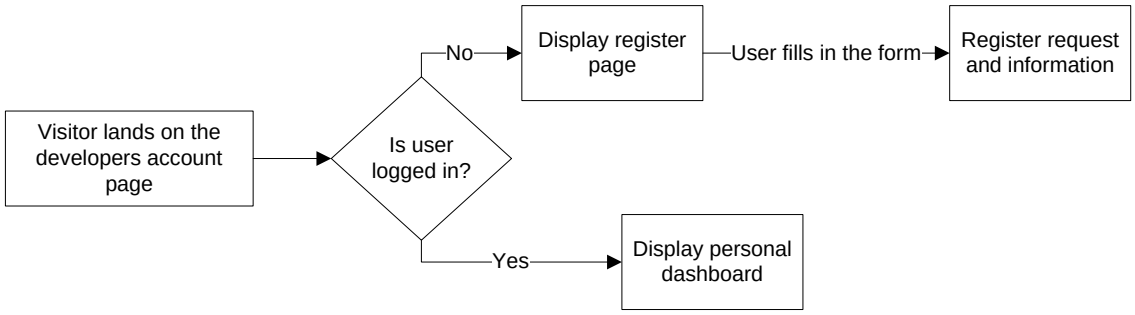

Figure 40 - Developer registration (user side)

Afterwards, administration will receive a notification, warning it of pending requests. At this point, the decision will be made on whether to accept the developer or deny its registration.

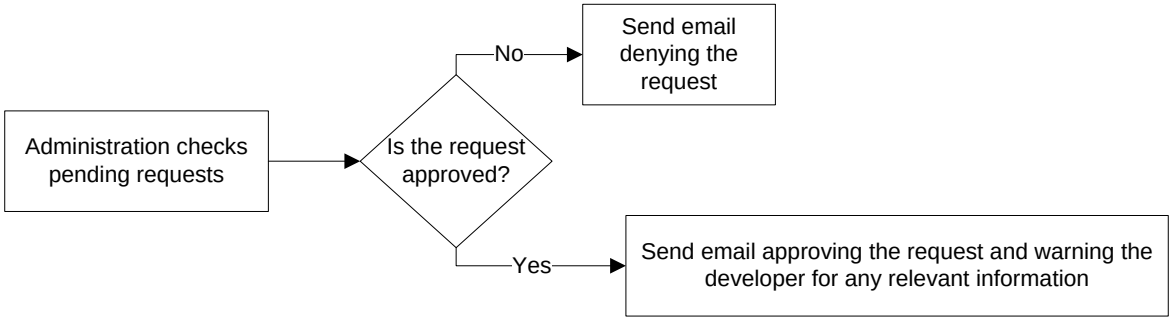

Figure 41 - Developer registration (administration side)

Although not an implemented feature, either event would result in the sending of an e-mail, welcoming the user and guiding it in its first steps or explaining why the registration request was denied.

4.2.3. **Application submission and approval**

Application submission and approval follow a procedure similar to any mobile marketplace submission like Google Play's, as seen in Figure 42.

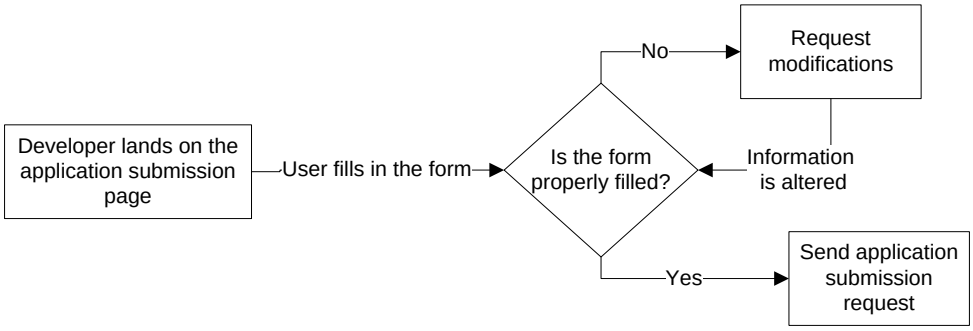

Figure 42 - Application publishing request

As developers enter a page of the platform dedicated to application submission, they are asked to fill in a form, writing the application's name, description and URL. After this has been properly filled, a notification is sent to administration, warning them of the pending request. After this, the administration is responsible for approving or rejecting an application based on its description.

4.2.4. **Application suspension**

Application suspension can come from failing to abide by terms of service. With administration watching for reported issues regarding applications and their usage of the network, they may choose to disable an application from using network resources via their dashboard. Administration must be able to identify the reported issue and uncover its cause. If it is breaching terms of service (and creates any of the behaviors specified in Figure 43), the administration may:

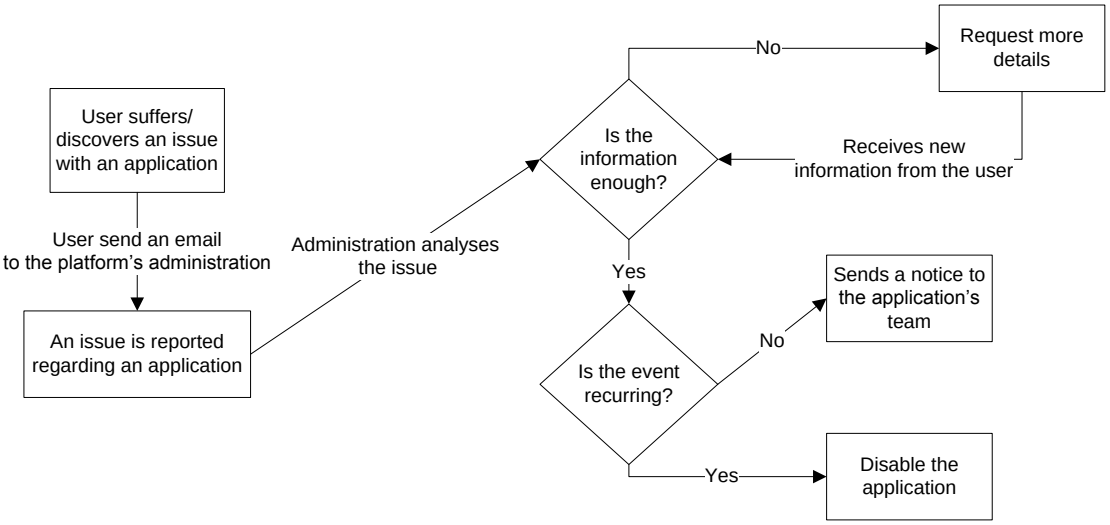

Figure 43 - Application suspension flow

a) Send a notice to the application's team, warning them of the reported issue and giving them time to correct it. The team may respond to the notice, justifying the problem and/or asking for more time, delaying the process – a decision that is left to the administrator;
b) If the issue is recurrent during the time-frame of the correction process, the user will be notified that the process is underway;
c) If the issue is recurrent then the decision of suspending the application is left to the administrator: if the events are scarce, a notice to the team should suffice; if, on the other hand, the problem is reported by a number of users, the application should be disabled.

The disabling of the application will be, effectively, revoking the given token. This will prevent the application from authenticating and will force the request of a new token.

4.3. **WebRTC**

WebRTC support comes by the integration of two crucial WIT components: the WebRTC-to-SIP gateway and WebRTC Communicator (WWC) which is composed of client and SDK. Being WIT Software products, these were not purpose-built, having modifications been made to fit WTE needs. WIT's WebRTC Communicator is a browser softphone, implemented using HTML5 and Javascript, supporting an array of communication features such as audio and video calls (using WebRTC) to SIP endpoints or other WWC users, chat messaging with file transferring and image sharing support. These are just two of the most relevant examples of the many features the client supports. The client itself is compliant with GSMA's Rich Communication specifications[76] and provides an SDK which can be shared among developers wishing to create applications supporting these types of telecommunication capabilities. The WWC connects to the WebRTC-to-SIP gateway, which itself is made in Java and connects to the network's P-CSCF to enable all the communication features.

4.3.1. **OpenIMS modifications**

Certain modifications had to be made to the IMS core in use, in order to support the WebRTC-to-SIP gateway and WWC client. Although this behavior is configurable, the gateway supports +351<number> contact formats and these number formats are forwarded to break-out servers in the OpenIMS core. Since no break-out server was integrated in the solution, the call would fail. As such, this call forwarding feature was disabled. Additionally, OpenIMS is very strict

on the format of the messages it accepts - if they are not "well-formatted", at least from a core perspective, they are automatically discarded. This happened when using the gateway and client – messages featured a port number next to the SIP address ("sip:name@domain:port"), not accepted by the server. As such, these addresses had to be cleaned in order to proceed.

4.3.2. **User interface**

In order to better understand how the features and modules are used, Figure 44 is shown. Here, one can see the clients chat interface, where several elements should be emphasized:

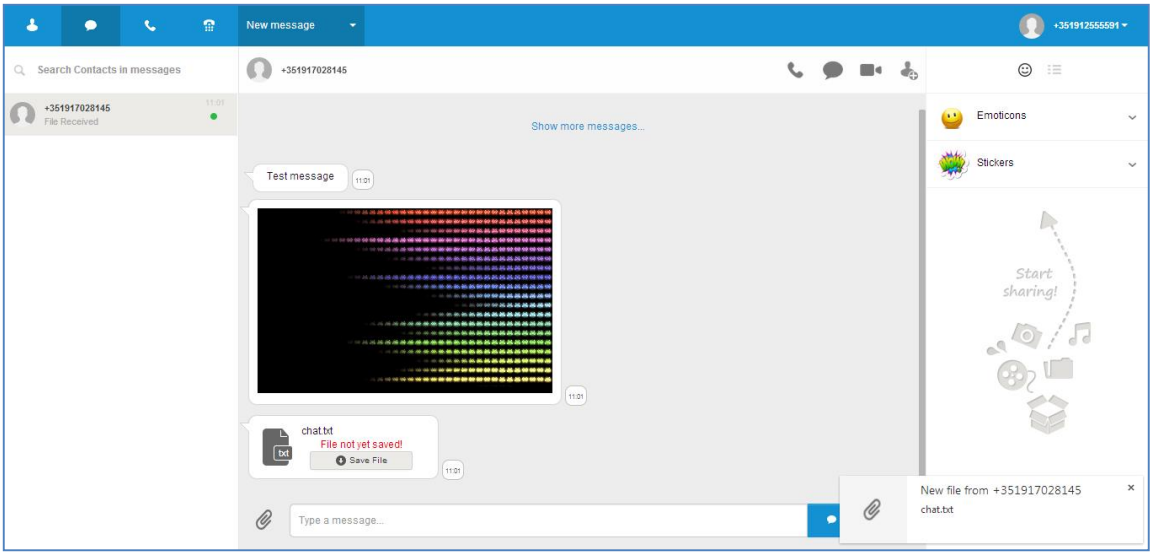

Figure 44 - WWC chat interface

- On the left, the users' address book is shown, where contacts are stored;
- Stickers and emoticons can be selected on the right, also being available by shortcuts in the chat fields;
- The chat timeline can be seen in the middle, with two files being represented – an image, shown as a thumbnail (may be seen entirely on click) and a file that can be immediately downloaded;
- On the bottom right, one can see a popup, generated when the client received the last message.

Some messaging features, however, had to be implemented specifically for the WTE solution, in order to support its messaging requirements. In order to process a SIP MESSAGE request made from the AS to the WWC, certain modifications had to be made to the client and SDK, since messaging was only originally available through Session-mode messaging (with a MSRP session established). These modifications, made in Javascript and affecting the processing of SIP MESSAGE requests, allowed the application to process plain text messages and rich messages (containing vCards and base-64 encoded JPEG image files), while maintaining chat messaging between clients.

4.3.2.1. **Plain text processing**

The SDK implements logic for addressing different SIP requests in a different manner. As such, modifications were made to the actions taken, by the system, upon receiving a SIP MESSAGE request. Whereas previously the client did not support pager-mode messaging, as referred above, these modifications now allow the client to process the textual content of the request, showing it in the clients' chat interface. This allows pager-mode and session-mode messaging to work seamlessly, from a users' perspective, a detail that is crucial when designing a clean, simple interface.

4.3.2.2. **Rich content processing**

A rich message, coming from the AS, will have a content-type such as "text/vcard" or "image/jpeg", containing the file's metadata which will either be textual (in the vCard case) or encoded (in the JPEG's case). As such, rich message processing followed a similar route to the processing done in the previous topic. In addition for interpreting their content, SIP MESSAGE requests are inspected for their content-type. Posterior processing depends on the result:

- If the content-type received is "text/vcard", the SDK will parse the metadata, retrieving a specific set of fields (name, telephone, email, workplace and location). Afterwards, an event, that would otherwise be a user action, is triggered – creation of a new contact in the clients' address book (Figure 45);
- If the request contains an "image/jpeg" content-type, the SDK will decode the content and show the image as a thumbnail in the clients' chat timeline, also making it available for zoom-in.

Both these actions allow for posterior download of the file sent and were both created specifically for this API use-case.

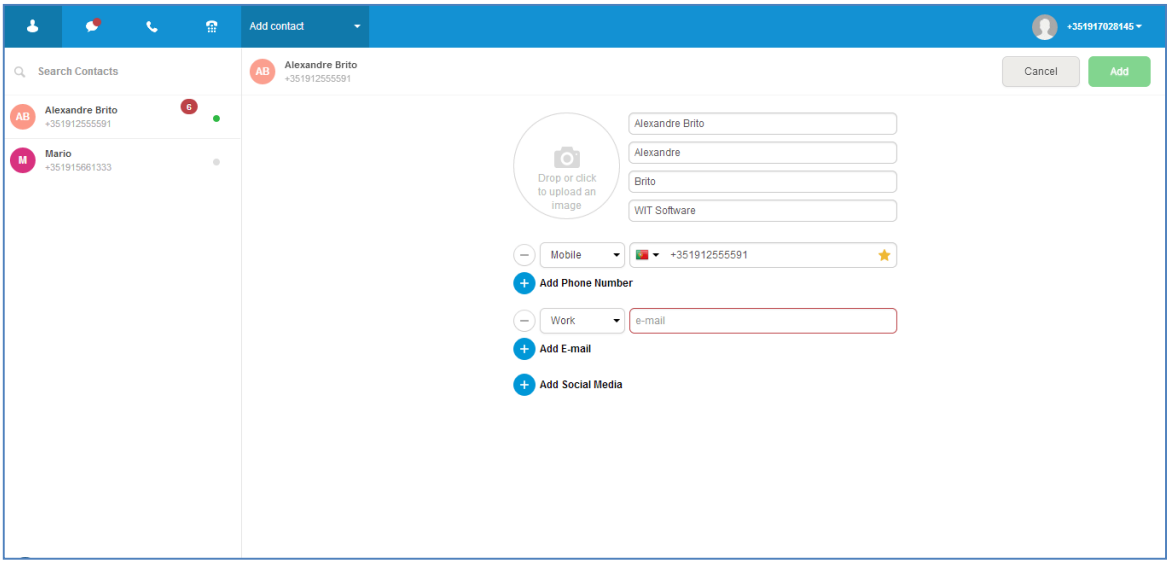

Figure 45 - Contact creation event on the WWC

4.4. Third party application

Again, taking on the reference architecture shown above on Figure 24, one can see several components that enable all the desired functionalities and the interactions between the third party applications and the network. An application will be typically composed of 2 core elements, as depicted in Figure 46 - a component that will handle third party call controls and one that will enable WebRTC features. A final, optional component is its own REST API for notification retrieval.

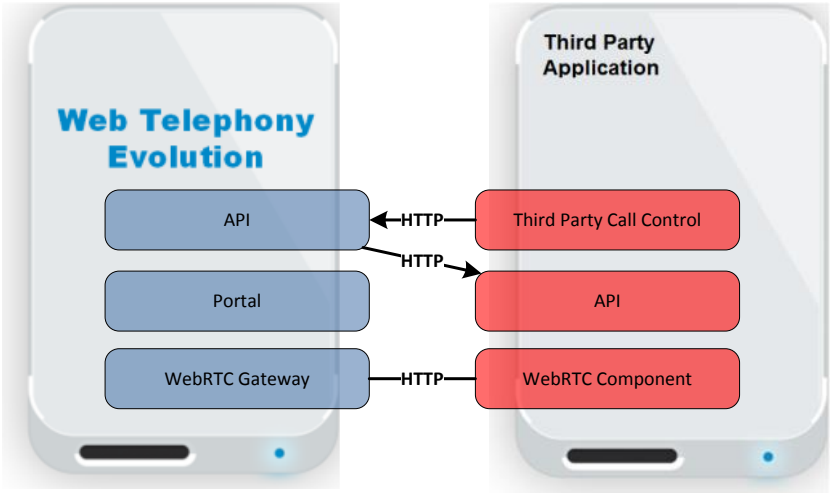

Figure 46 - Third party application architecture

All components on Figure 46 have their own very distinct responsibilities:

- The 3PCC (Third Party Call Control) component will be responsible for sending requests to the solution's API, triggering all the desired events – from call placement to messaging;
- For first party call controls, the WWC will be used, communicating with the WebRTC-to-SIP gateway. This is essentially a client, with a proprietary SDK used to develop the application. Again, these components were not developed in this solution, although certain features in the SDK and client were implemented in order to create this dissertation's proof of concept (the pager-mode messaging features via SIP MESSAGE);
- Finally, the application's API will have the sole responsibility of receiving notifications from the AS. If such is not implemented, a mechanism for long-polling is also available for applications to retrieve incoming notifications.

This architecture is for an example application nonetheless; if, for example, the developer wishes to only focus on a web application that can trigger 3PCC functionalities, the WebRTC component would not be used.

4.5. **Chapter summary**

Throughout the implementation phase, as features gained form and with the decision of using VaaS as the baseline and the OneAPI as reference API model, certain changes to the requirements were made to the AS. Almost all of the planned features were implemented, from call control to notifications and authorization/authentication (user/application authentication and authorization was completed). Conference calling is also implemented at this stage, as well as a long-polling mechanism, giving a developer a fallback mechanism for notification retrieval, enriching the application and possible use-cases. Developer authentication and the retrieval of application descriptions were first planned but posteriorly discarded. API statistics, billing and presence services also remain unimplemented at this stage. This is due to their priority, with focus being deviated to more important functionalities.

Additionally, certain limitations should be noted. With VaaS' media layer relying on deprecated methods and with the server only supporting the G711 codec at this time, errors may occur if the configured call hold music does not follow very specific conditions – encoded in G711, with 16 bits and 8000Hz. Errors take the form of internal errors, with media not being played, or reproductions at very low sound levels.

Signalling wise, there are also some limitations that exist, were if one client disconnects the BYE message is not forwarded to the other parties. This decision was made due to the OneAPI specification of appending the call to the server, having been decided that the server would only maintain a B2BUA stance and not proxy requests.

Focusing on the portal, as referred before, most of the features planned were implemented, as well as the features for the WWC client. This client is also able to display vCard handling in a manner not planned initially, which is seen as a positive decision in terms of user experience (presented in 4.3.2.2), contributing to the WWC project.

In short, the decision of using VaaS, the WebRTC-to-SIP gateway and WWC client for bootstrapping the solution is seen as a positive one, resulting in a more advanced, stable and feature-rich proof-of-concept, with room for evolution.

Chapter 5

Evaluation and results

A software evaluation strategy will detail, from a high-level perspective, how testing will be approached during this dissertation's development, including what types of tests should be done and their priorities, alongside the different levels of testing that have been planned. These tests will be used to study several system characteristics, such as performance under load, delays, mouth-to-ear delay, among many others metrics. System performance is crucial and will affect how well the solution responds to requests made, affecting overall quality of service (QoS), from call setup times to the delivery of messages in due time. As such, the system must be able to handle increasing workloads, all the while maintaining a stable environment for user interactions. With this concern in mind, the network components in use and the architecture planned (from a high-level view) must be analyzed, identifying possible bottlenecks which could eventually limit the amount of calls that the system would be able to handle. Its performance will be influenced by different details, such as the implementation of the components themselves.

System monitoring is also a concern that should be addressed. Performance indicators (KPI, short for Key Performance Indicators), could be used to monitor the system and make scalability decisions – one could implement a mechanism that, upon monitoring an above average load on the system, would trigger the deployment of another node, resulting in better load-balancing of incoming events. Additionally, and from a call perspective, analytics could be made available to the clients allowing them to dynamically renegotiate codecs when the quality of a call degrades. Parallel to these indicators (events for second, bandwidth, among many others), are aggregate metrics such as Calls Per Second (CpS). Due to softphone to WebRTC calls possibly needing to go through transcoding, performance and call quality might be affected in these scenarios, due to the processing demands required. To analyze this issue, mouth-to-ear delay tests were made in addition to load-testing studies to assess the solution's performance.

Load-testing will be used to simulate multiple and concurrent access to the solution's features by users. This testing is designed to ensure that the system is able to cope with numerous concurrent calls in an efficient manner, consequently testing its scalability and how it behaves under load. In this testing scenario, care should be taken into analyzing how delay increases the setup of new calls sessions and if requests are lost. Bottlenecks in the system could also be identified - for instance, if the API becomes a bottleneck or if the solution's SIP layer is not able to cope with incoming requests. To this end, multiple tools may be used, such as MTS[77] or

SIPp[78] which help load-test the platform by generating SIP traffic and The Grinder[79] or Vegeta[80], used to load-test the API. These types of test cases are of great importance to this dissertation, providing feedback on its quality and possible weaknesses that should be promptly mitigated.

On the other hand, mouth-to-ear tests are designed to assess performance on an audio transmissions level, measuring the time it takes for a callee to listen to the words spoken by a caller – higher delays lead to client-side difficulties, with callers losing the sense of real-time communication as audio gets delayed across the network. To test this scenario, an approach will be defined using the Audacity audio tool[81], which enables audio manipulation to fit a number of needs. With this test-case, the components which add more delay to communication will be identified, measuring how they impact the platform and if the platform itself is efficient enough to fit this dissertation's needs.

5.1. **Testbed description**

This topic provides an overview on the hardware components used in the server and client machines, as well as software versions in use and possible component configuration. Although a full description may be seen in annex F), a few of the main characteristics will now be described. The server machine, where IMS core, AS and gateway will be deployed, has an Intel Pentium dual core processor with three gigabytes of memory, running an Ubuntu operating system. The client machine, where test clients will be deployed, has a more powerful Intel Pentium dual core processor with eight gigabytes of memory, running a Windows 7 operating system. Both machines have physical connections established to a local network, diminishing external constraints that might influence the results.

Due to restrictions on the hardware currently available, the following components were executed in the server machine:

- OpenIMS core (R1186) and Sippy RTPproxy (v1.2.1), along with all the databases;
- Apache Tomcat (v7.0.29) and Mobicents (v2.0.0), using Sun's JVM (v1.6.0_45) with no default configurations changed;
- AS (R3603) and WebRTC-to-SIP gateway (v3.7).

Since the test machine had enough physical memory to handle the components, this was not regarded as a bottleneck. OpenIMS modifications also had to be made, disabling logging and increasing child processes available to the components, to maximize server throughput (annex G)

contains such modifications). The reasoning behind this decision lies on logging being a very expensive writing operation, which greatly affects performance.

The SIP testing tool used was MTS (Multi-protocol Test Suite[77]), with the test scripts developed following the asynchronous model described in its users' manual[82]. The software ran on Sun's JVM configured with one gigabyte of memory and several default proprieties were modified from their default values (seen in the H) annex). MTS is a very feature rich tool, able to simulate SIP requests and RTP flows, providing metrics and graphs relating to the gathered data, display request/response delays, transaction duration, among many others. However, it requires a substantial amount of processing power and a particular bug was noticed during the tests: in certain cases, SIP transactions were not correctly closed, resulting in statistics revealing missing BYE's. To mitigate this issue, a customized Python script was used to parse the logs and confirm the results.

5.2. **Call throughput**

The objective of these tests was to evaluate the performance of the solution handling common operations performed by endpoints, focusing on calls only. Firstly, a strict IMS-only signalling test was made, to establish a baseline with a scenario based on call invites. The second test was based on the previous scenario, but with media exchange enabled via a media proxy, increasing performance requirements to handle all transmissions. In the third test-case, the media proxy was discarded and the AS was used, a scenario design to test the AS's ability to handle intensive and concurrent user actions. The fourth and final test-case is based around the WebRTC support, testing the impact it has on the platform.

Across all tests, certain default parameters were established to ensure consistency in the results. Before conducting the tests, clients were previously registered in the network and call establishment was set to 3 seconds, while call duration was set to 30 seconds. Therefore, the maximum number of simultaneous calls (SC) in the tests can be calculated as $SC = 33 * CPS$ (Calls Per Second, the rate at which the scripts will generate new calls).

5.2.1. **IMS signalling**

This test was made with only the IMS core deployed on the server, aiming to understand the signalling capacity of the core. Baseline values were established and will be used for comparing the results gathered when the other components are involved in the communication.

Call parties (both "callers" and "callees") were simulated using MTS scripts. Two users were created on the core's HSS and registered before each test with an expiration interval larger than the test duration. The number of clients connected is not relevant, since having the clients registered is virtually inexpensive and call establishment is not impacted by having several calls between the same clients. For each test, the scripts executed the steps in Figure 47 continuously.

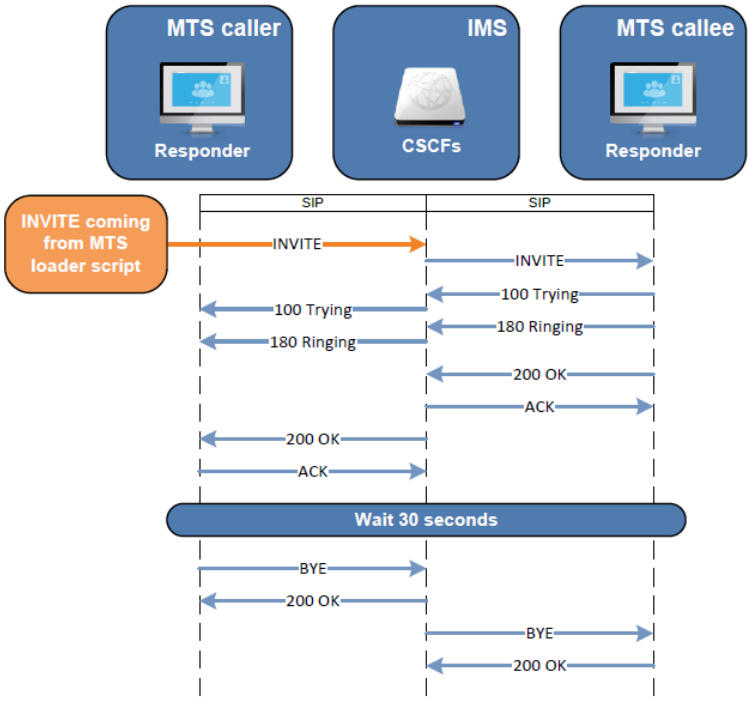

Figure 47 - Default script steps and signalling flow

The test scripts were executed multiple times, with different CPS, revealing how the server behaved when faced with theses variations. Figure 48 - Figure 51 reveal how the solution behaved, in terms of response delay, to a 165 to 660 simultaneous call variation. These delays (measured between the sending of an INVITE and receiving of a 200 OK) are gathered and reported by MTS, with each graph corresponding to the results of an individual test (chosen for reflecting an average scenario after a certain amount of testing).

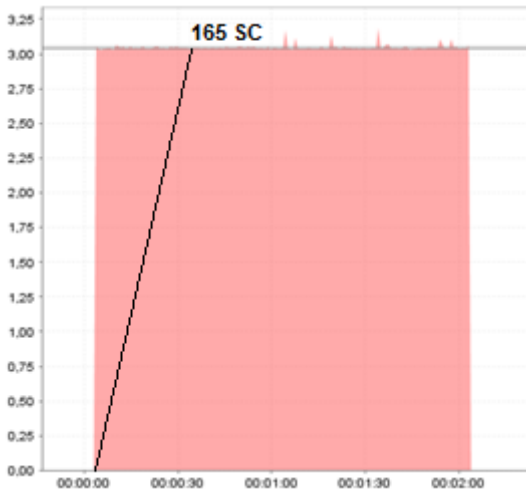

Figure 48 - Call establishment times at 165 SC (signalling only)

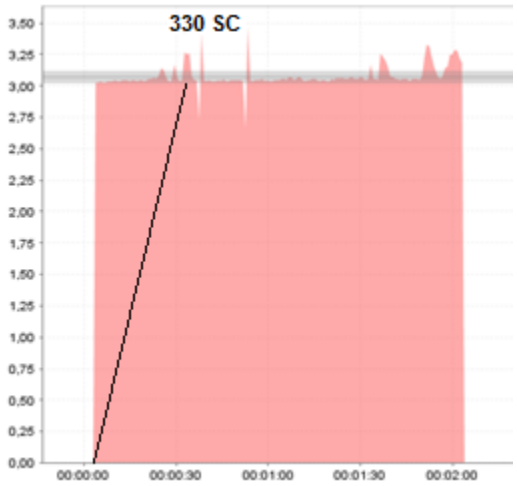

Figure 49 - Call establishment times at 330 CS (signalling only)

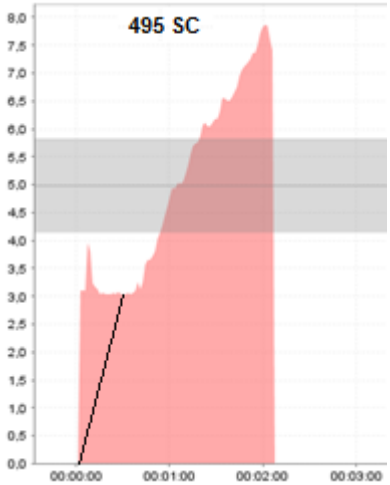

Figure 50 - Call establishment times at 495 SC (signalling only)

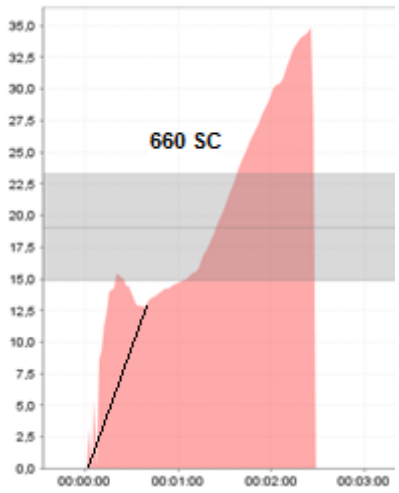

Figure 51 - Call establishment times at 660 SC (signalling only)

Response times cannot be lower than 3 seconds, since this is the delay introduced between 'Ringing' and 'Established' states. As such, the added delay at this call rate is considered acceptable. Increasing SC value results in growing response times and, consequently, delay - for instance, at 660 SC the increase in average delay is ~15.95 seconds (from 3.05 to 19 seconds).

Additionally, certain spikes can be seen in the delay, increasing and especially noticeable in the end of Figure 49 and beginning of Figure 51. These are attributed to the allocation of more memory by MTS, as the software demands more resources in order to process all the information. Although delays increase as SC values become higher, as long as the delay remained lower than the request timeout time, calls would still be able to be established, even though this is an unacceptable scenario for client communication purposes (due to the increasing call setup time).

At its best, the IMS core was able to handle 330 SC without errors for 2min, for an approximated total of 1200 calls made. Response times were overall stable during the entire test duration, as were CPU load and memory occupation. Additionally, at this level, the core was able to handle all requests in the same approximate delay, as represented by the stronger bar in Figure 52.

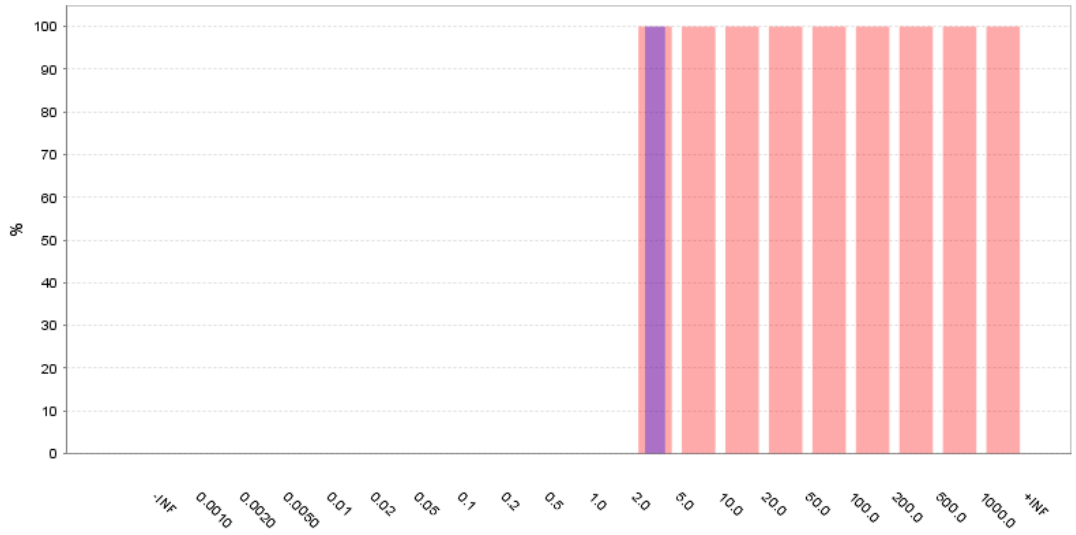

Figure 52 - Call establishment time distribution, in seconds (330 SC, signalling only)

One can see a remarkable difference it constitutes when comparing these results to the results in Figure 53, with the rate set to 660 SC.

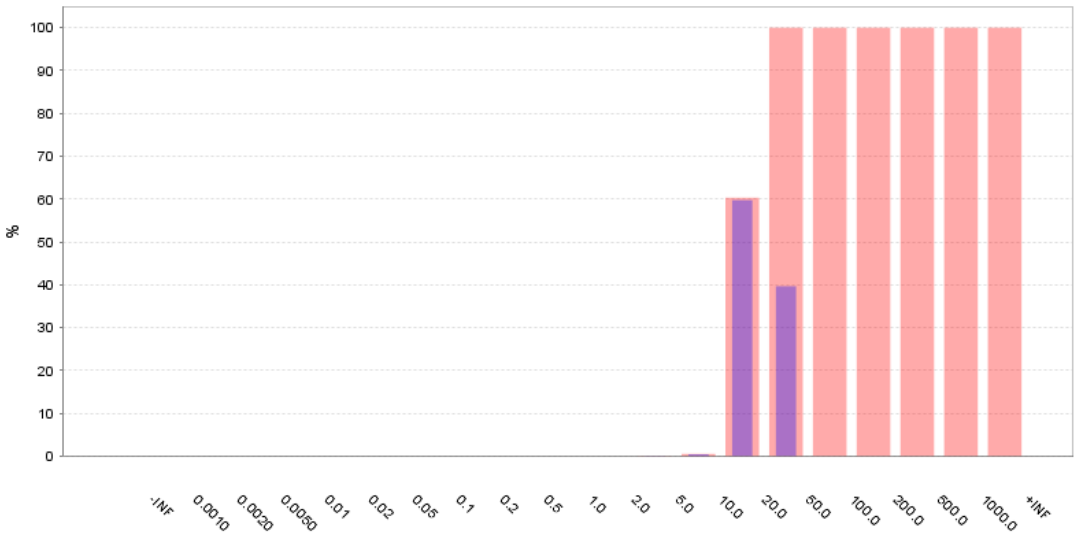

Figure 53 - Call establishment time distribution, in seconds (660 SC, signalling only)

Whereas in the first, all requests where responded in the same approximate delay, here 60% where answered in an average ~19 seconds and 40% with an average delay of ~25 seconds. This is caused by increased MTS processing time and effort to deal with all the transactions, due to the amount of messages received server side, making it the bottleneck in this scenario.

5.2.2. **IMS signalling and RTP exchange**

More than just signalling, audio calls involve (among others) RTP exchange and possible communication with networks AS's. Thus, before testing API or WebRTC performance, different test sets involving media exchange or AS deployment should also be executed, since both may hinder server performance. However, both these tests are not applicable in the current scenario. Firstly, the core's P-CSCF does not relay media, serving merely as a signalling proxy. As such, media is exchanged in a peer-to-peer manner, not adding load to the core. Secondly, since the AS triggers its own events, initiating the sessions and remaining on the signalling path, the core does not interact with it in first-party calls. Simply deploying the AS on the server will only consume memory, not interfering with the server's performance – incoming requests will not be forwarded to the AS. In order to properly define a baseline performance value for the first case, a different approach must be taken. The AS will act as a media bridge when tested, with parties establishing a media stream with the server and exchanging data. As such, a similar scenario will be mapped in this case, deploying an RTP proxy (Sippy RTPproxy[83]) which will act as a media bridge for all communications. This will generate the flow in Figure 54, testing signalling and media handling performance.

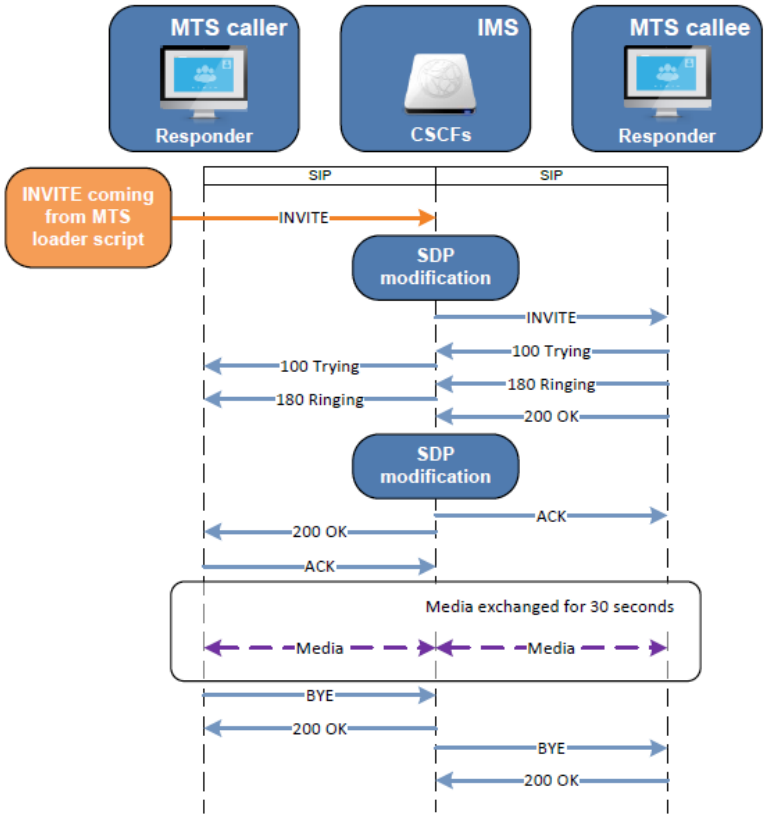

Figure 54 - RTPproxy signalling flow

An error was uncovered in the P-CSCF when deploying the RTPproxy and enabling its usage. During calls, the RTP ports exchanged in the SDP would be changed to those allocated by the proxy, but the IP would remain the same, breaking communication. The issue uncovered and the modification applied can be seen in annex I). Having fixed the issue, the P-CSCF began to properly change the SDP to the IP coming from the RTPproxy, enabling media relay. For this test case, the MTS tool simulated a G711 stream, with the encoded audio consisting of packets with a 160 bytes payload sent every 20ms (50 packets/s). As in the scenario above, testing was executed multiple times, with Figure 55 - Figure 58 revealing how the solution behaved with different SC rates.

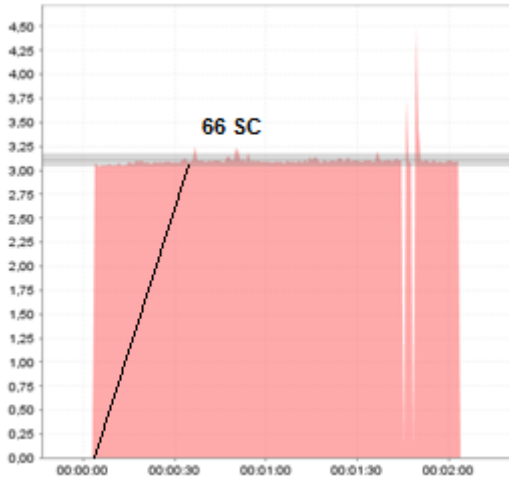

Figure 55 - Call establishment times at 66 SC (signalling and RTP)

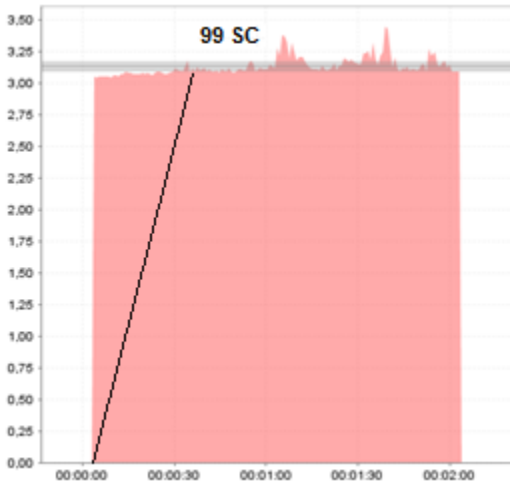

Figure 56 - Call establishment times at 99 SC (signalling and RTP)

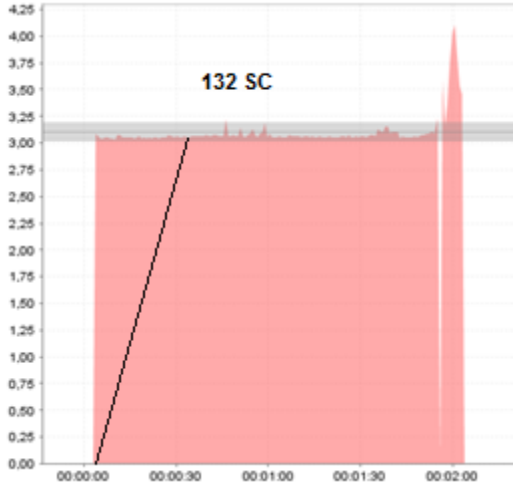

Figure 57 - Call establishment times at 132 SC (signalling and RTP)

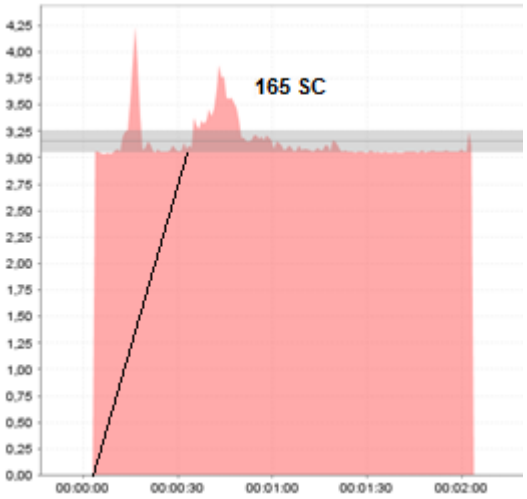

Figure 58 - Call establishment times at 165 SC (signalling and RTP)

Analyzing Figure 55 - Figure 58, one can see similarities between them, as they show that average response delay is maintained at ~3.15 seconds. However, this does not reflect reality since, above a rate of 99 SC, the test reveals that calls are lost – 0.33% of the calls at 165 SC, 1.02% at 198 SC and increasing. Delay may remain similar, but requests are lost. This result is unacceptable for a communication system, where reliability is a key factor. The reason for the lower rate and calls lost lies on the MTS, which cannot handle all the signalling and media streams, due to hardware limitations (as seen in Figure 59 and Figure 60).

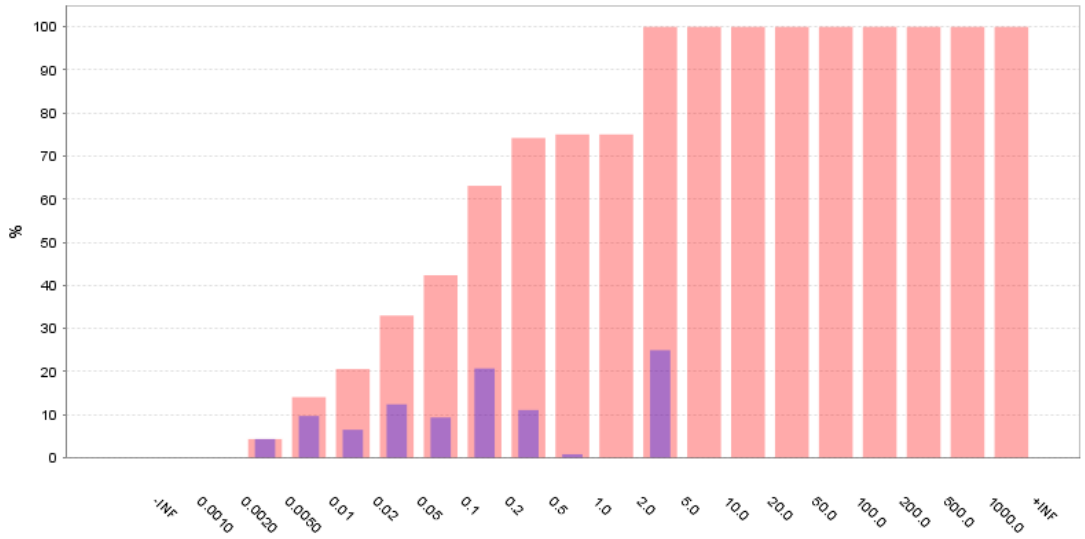

Figure 59 - MTS response time, in seconds, at 99 SC

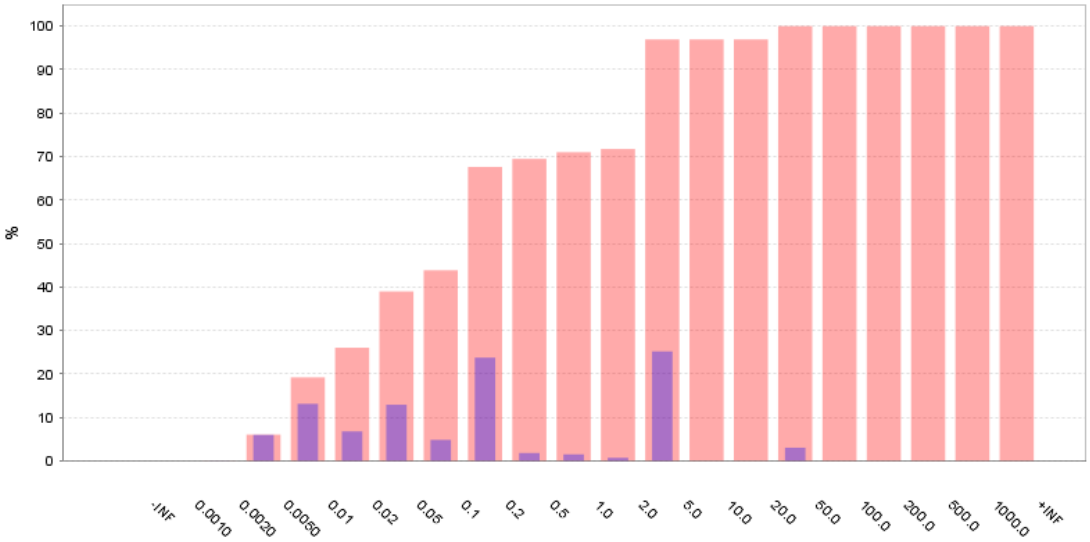

Figure 60 - MTS response time, in seconds, at 132 SC

As such, 99 SC was established as the baseline rate the server can handle, resulting in 360 calls made in 2 minutes. Again, at this rate, the server was able to handle all requests in the same timeframe, as seen in Figure 61. Even though some delay spikes may be seen in this figure, similarly to the previous test case, this was seen as not affecting performance at this CPS rate. Again, the cause of these spikes is MTS, with Java acquiring more resources from the system.

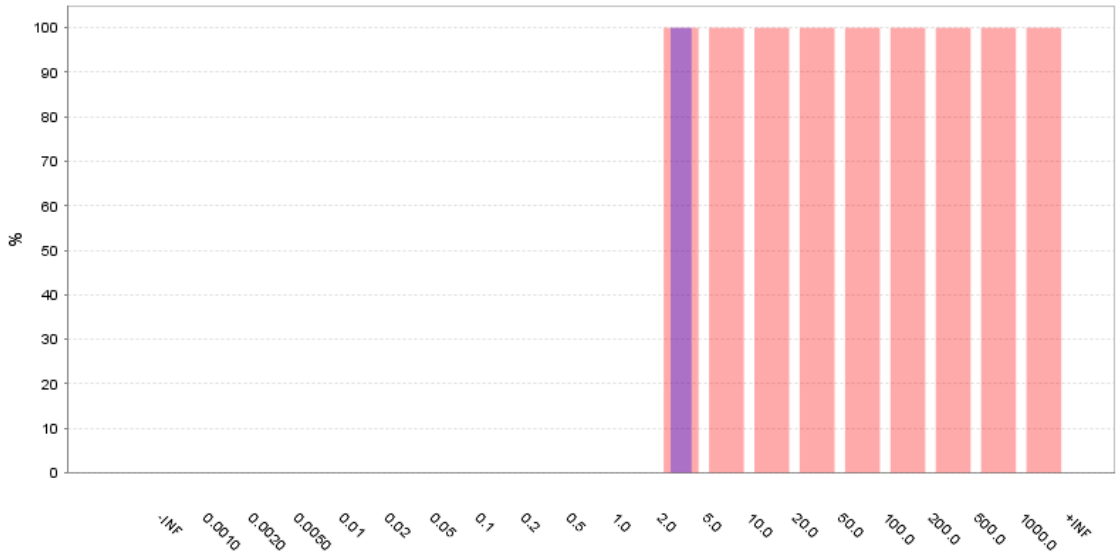

Figure 61 - Call establishment time distribution, in seconds (99 SC, signalling and RTP)

5.2.3. **AS**

This test set was made with the IMS core and AS deployed on the server, with calls being triggered by the API to registered users. Similarly to the previous test-case, resulting values will be compared with the baseline established in the previous test scenarios, analyzing how server performance as been affected.

Again, test clients ("MTS Responders", as referred in Figure 62) were simulated using MTS scripts, with default values applied (call establishment was set to 3 seconds and call duration to 30 seconds). The tool used to stress test the API, represented by "Loader" in the same figure, is Vegeta[80]. A very simple tool to use, it provides a number of interesting functionalities, such as request rate and test duration parameterization, with methods for retrieval of delay metrics, among many others.

Again, two users were created on the core's HSS and registered before each test with an expiration interval larger than the test duration. For each user, the scripts executed the steps in Figure 62 continuously.

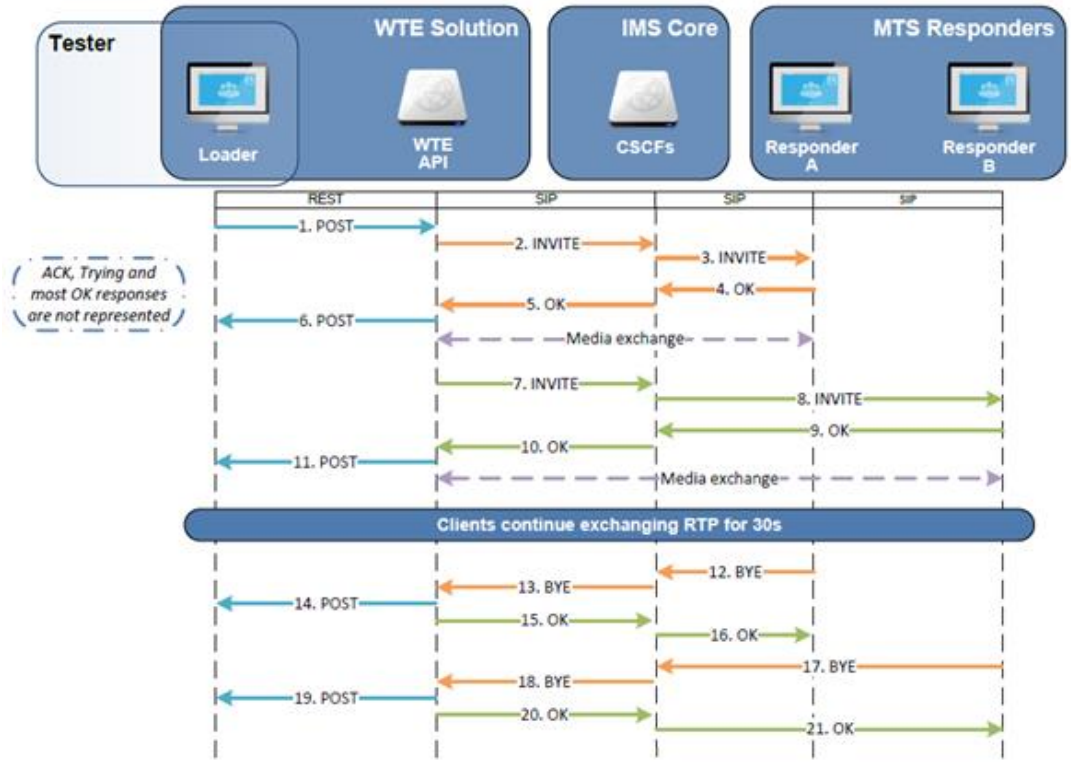

Figure 62 - API signalling flow

Similar to the previous test-case, upon INVITE acceptance by the MTS script, a G711 stream is generated, following the same parameters: encoded audio consisting of packets with a 160 bytes payload sent at a rate of 50 packets/s. The test was executed and again, the results gathered indicate that MTS was the bottleneck. When going from 99 SC to 132 SC, the delay increase was such that calls were lost, as depicted in Figure 63 and Figure 64.

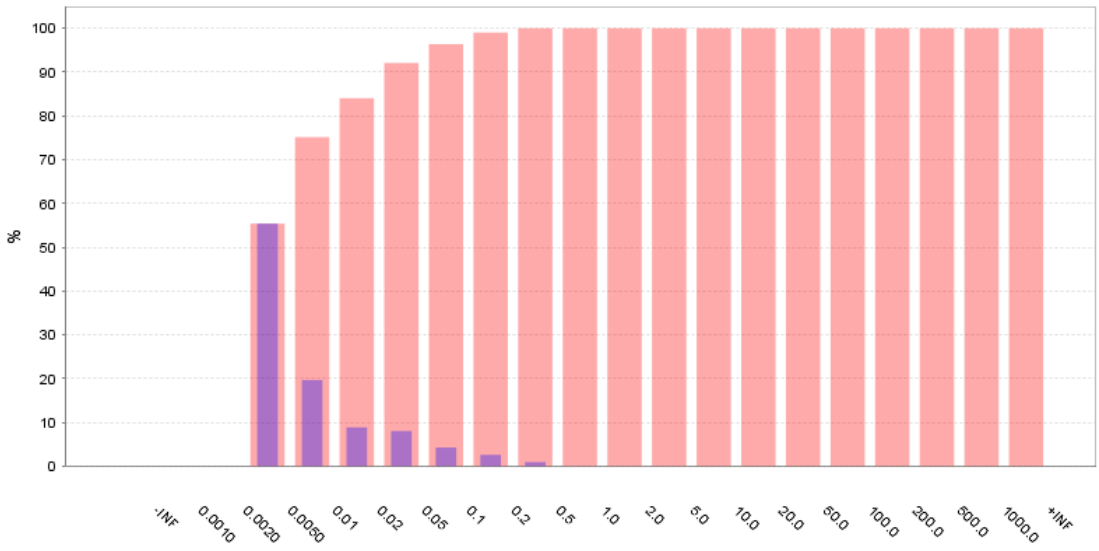

Figure 63 - MTS response time, in seconds, at 99 SC

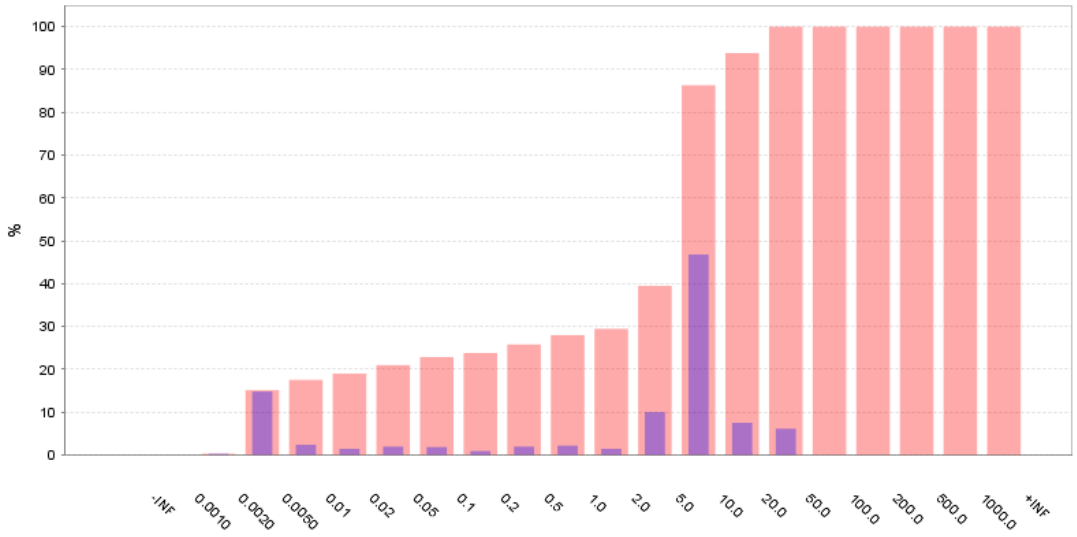

Figure 64 - MTS response time, in seconds, at 132 SC

The MTS approach taken thus far could not be replicated in this scenario, since INVITEs were not being made by the MTS tool and responders were not talking directly to each other (establishing a connection to the server in this case). Consequently, delay graphs like the ones shown before could not be generated. In order to mitigate this issue, logging was enabled in the AS and additional log messages (such as the one depicted in Snippet 3) were implemented when sending an invite and receiving a response. The first three sections of the message are not relevant for these results, but the rest of the message reveals all necessary information to calculate signalling delays, divided by hyphens - the first section ("Invite sent") reveals the SIP request's method, the second the call identifier associated with the request and the third a timestamp down to the microsecond. Logging was then captured, stored in a separate file and a custom python script was made to parse the file and calculate minimum, maximum and average delays of the featured calls (represented in microseconds).

```
2014-05-27 11:18:00|ERROR|                |DefaultSipServlet              |Invite
sent - 0a68c91557f0b323eb7f594c4e921c33@127.0.0.1 - 2014-05-27 11:18:00.621
```

Snippet 3 - Call logging in the AS

At 99 SC, the results were as follows: for a total number of 720 calls, the mean average delay was 138 ms (from a minimum of 115 ms to a maximum of 506 ms). The delay cannot be inferior to 0.1 seconds (or 100 ms), since this is the waiting period established by the MTS tool to respond to an INVITE request with a TRYING response. When comparing this to results seen before, in Figure 56, one can see the similarities of the results, as at a similar level of load, the AS behaves much like the RTPproxy.

API response metrics such as mean and maximum delay were also retrieved from Vegeta after the tests, showing an average latency of 28 ms at 99 SC (to a maximum of 740 ms) and an average latency of 45 ms at 132 SC (to a maximum of 1.25 seconds).

In short, although the API may have been slower at times to respond to incoming Vegeta requests, signalling behaved correctly. More tests could have been made if more powerful hardware was available for the testing machine.

5.2.4. **WebRTC and AS**

One final test-case should be made to the WebRTC support, using the gateway and WWC client in conjunction with the AS, to see how the different interactions affect platform performance. Although the WebRTC components were not developed for this dissertation, an end-to-end stress test is relevant due to the interactions made between the WWC client, gateway and IMS network.

By itself, the WebRTC support will not put extra-strain on the AS, since it would only see endpoints - the impact a WebRTC client makes on the AS is no more significant than any other SIP endpoint. However, more memory and processing power would be used in the server, since the gateway would also be deployed, alongside all the other components. Demands such as transcoding are non-existent in this case, lowering processing demands: the AS only supports the G711 codec, which is also supported by the browser itself (Chrome supports this codec along with OPUS and VP8, among many others)[84]. Load-testing the WebRTC support will take a different approach from the one taken thus far, depicted in Figure 65, since several conditions are in place – from WebRTC needing browser capabilities for media device usage to user actions needing to be mimicked in other to produce automatic tests (client heavily rely on user interactions).

In order to mimic the user actions needed to answer calls, a purposed built WWC test client will be used, registering N different clients in the IMS to answer all incoming calls. This test client will run on the client's machines browser, thus enabling the capture of local media. An obstacle may immediately be envisioned, with the browser very likely becoming a bottleneck, will all the media connection established and demanded resources. A similar workflow as the one in the previous test-scenarios will be recreated: MTS will be responsible for triggering API calls to an answering MTS script and to the WWC client, placing N calls. MTS will be used instead of Vegeta, due to the need to dynamically change the requests made to the API to call each of the registered WWC users.

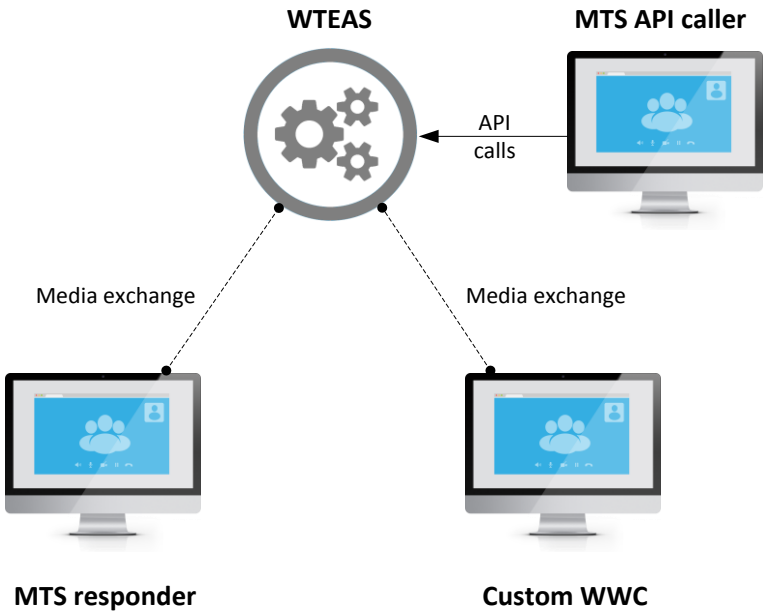

Figure 65 - WebRTC load-test approach

Load-testing the solution in this scenario was made and, at first, the browser became a bottleneck at around 20 concurrent calls, with call acceptance being delayed until other ongoing calls were terminated. Not an unforeseen obstacle, the chosen approach had to be tweaked due to Chrome's limitations, with clients being executed from two different Chrome browsers to increase the number of concurrent calls (due to hardware limitations, more clients were unavailable). The same test was done re-done, with only 10 clients being registered in each browser, and the results achieved reflect what has been seen thus far.

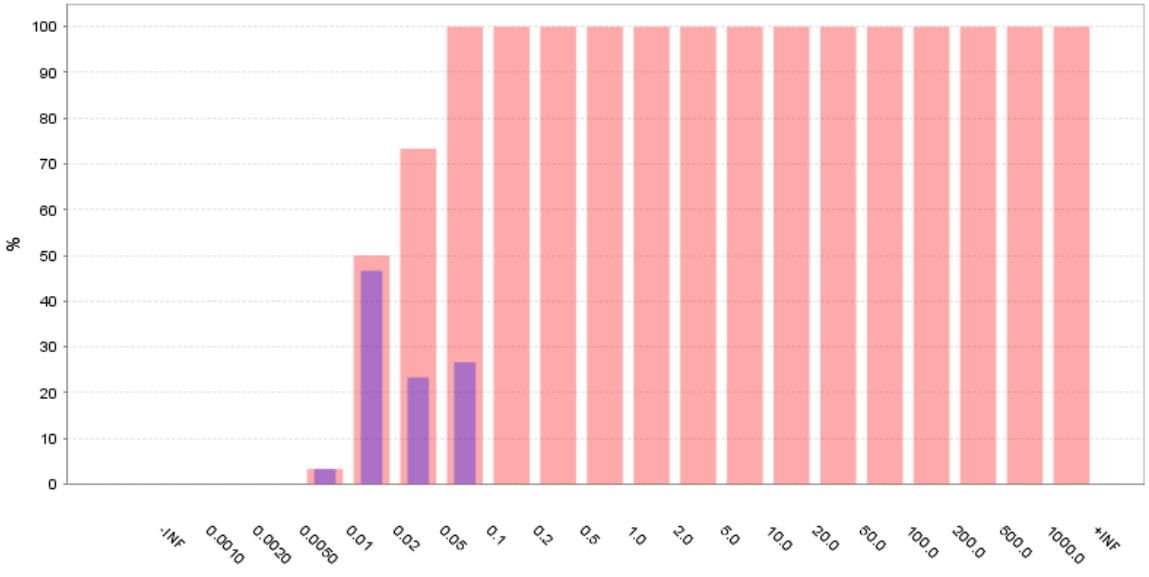

Figure 66 - MTS response times (20 concurrent calls)

As shown in Figure 66, even though MTS response times seem low, they are increasing rapidly, reflecting what has been seen thus far across the previous test-cases. Although SIP exchange delays remain in very satisfying intervals at this level, this increase already gives the indication that MTS will most likely become a bottleneck in more intense scenarios. With hardware limitations forbidding higher call rates and the analysis made, the SC value considered for this case will be 99 SC, similarly to the previous test.

5.2.5. **Test results summary**

Due to the restrictions imposed during this test phase, in terms of hardware limitations, more intense performance testing was not made – higher simultaneous calls or test durations would be factors that might wield more in-depth results. Although not reflecting the components real capabilities and not allowing the search for more accurate bottlenecks, the tests made already feature a level of insight to the components behaviour under load, abiding by the non-functional constraints referred in the beginning of the chapter.

Table 2 resumes the findings of the performance tests, expressed in number of simultaneous calls.

Test	Results
IMS signalling	330
IMS signalling and RTP	99
AS	99
WebRTC and AS	99

Table 2 - Performance test results summary

5.3. **Mouth-to-ear delay**

In addition to load-testing, inquiring whether performance becomes an issue under substantial amounts of user actions, mouth-to-ear delays were also analyzed in the different scenarios. This call quality metric reveals voice latency within a call, i.e., the timeframe between a word being spoken on one end of the call and being heard on the other. A multitude of factors may interfere with this metric, from network quality to server/client implementation and audio conversion.

Using a tool called Audacity[81], with a configuration inspired by articles where similar test-cases were done[85] and its own manual, the default testing approach depicted in Figure 67 was followed.

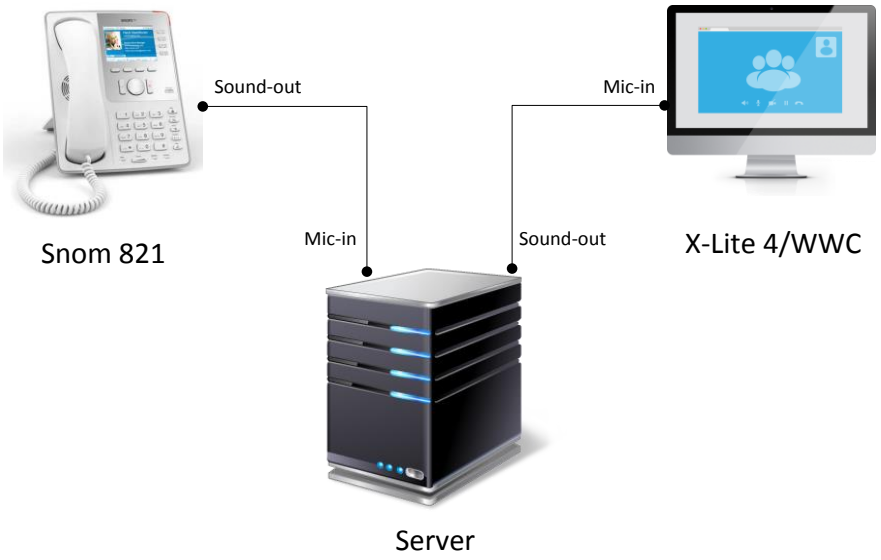

Figure 67 – Mouth-to-ear test configuration

In this approach, Audacity is executed in the server machine, alongside all other network components (OpenIMS, WebRTC-to-SIP gateway and RTPproxy, among others). Two connections were established between the server and both caller/callee:

- One where the server's headphone jack and the caller's microphone jack (either X-Lite 4 or, in WebRTC scenarios, WWC) were connected via a male-to-male line;
- A second one, where the callee's (Snom 821) sound-out was connected to the server's line-in. This was done by taking a normal 4P4C cable, that would otherwise connect a headset to the telephone system, and soldering both microphone and sound lines to two separate headphone jacks (mono, 3.5 mm).

To start the test, a call would first be established between both parties. Audacity then played a pre-recorded media file (containing a total of 31 sound pikes) which would feed the caller's microphone and, in parallel, also record the sound being received by the callee. This would enable a direct comparison between the time a sound was made, in the media playback, and the time the same sound was received.

Following the same analysis featured in the performance testing, the first test made was a peer-to-peer call, establishing a baseline for comparison, with posterior tests incrementally adding solution components to see on these impacted audio transmissions.

5.3.1. **Peer-to-peer call**

This test, made with only the IMS core deployed on the server, serves as a baseline for average delay between client calls (Figure 68).

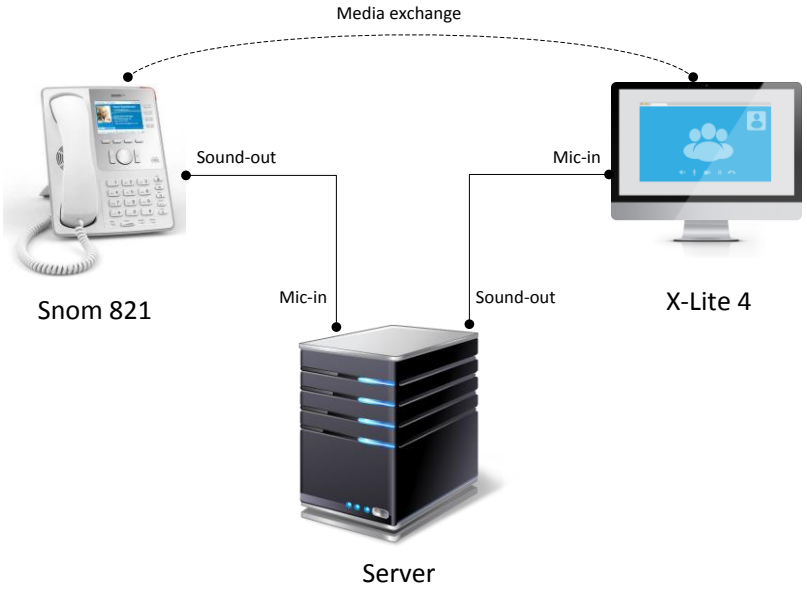

Figure 68 - Peer-to-peer test configuration

After being fully established (both clients have accepted the call), the actions referred above were executed – media playback to X-Lite's microphone, in parallel to Snom microphone recording. This test scenario led to the following results:

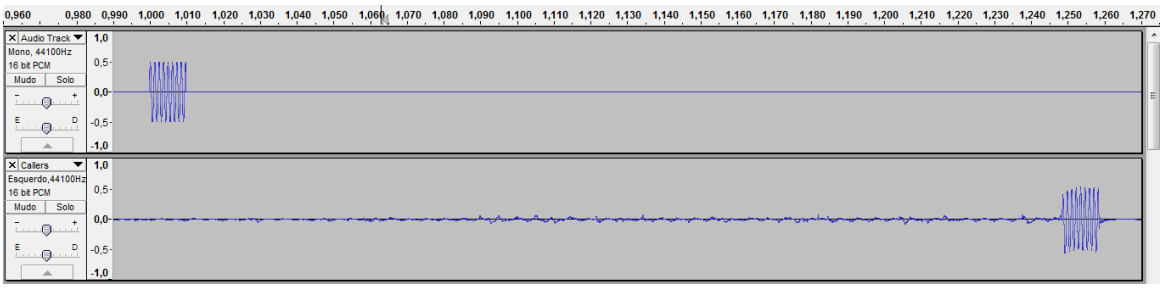

Figure 69 - Peer-to-peer audio comparison

The top axis in Figure 69 represents time in 5 ms increments, with the original media file being the audio track on top and the recording at the bottom. As one can see, the first spike in the original track began at approximately 1 second, whereas the first spike in the recording only began at 1.25 seconds – a difference of approximately 250 milliseconds. This is a very significant delay which, according to ITU's recommendation on one-way transmission times[86], may result in only a satisfactory call experience, falling under a "Users satisfied" level of acceptance – ITU recommends a delay of 150 ms and establishes an upper bound at 400 ms. This 250 ms delay may be a result of the network configuration or bandwidth, codec delays or sound acquirement by the devices, but is unrelated to any of the WTE solution components, since these were not deployed nor used in this test. As such, this test result will be used as a baseline in the following tests.

5.3.2. **RTPproxy call**

After deploying the RTPproxy in the server, this test will now trigger a significantly different flow from the one above, with media being relayed by this proxy to the call participants, as represented in Figure 70.

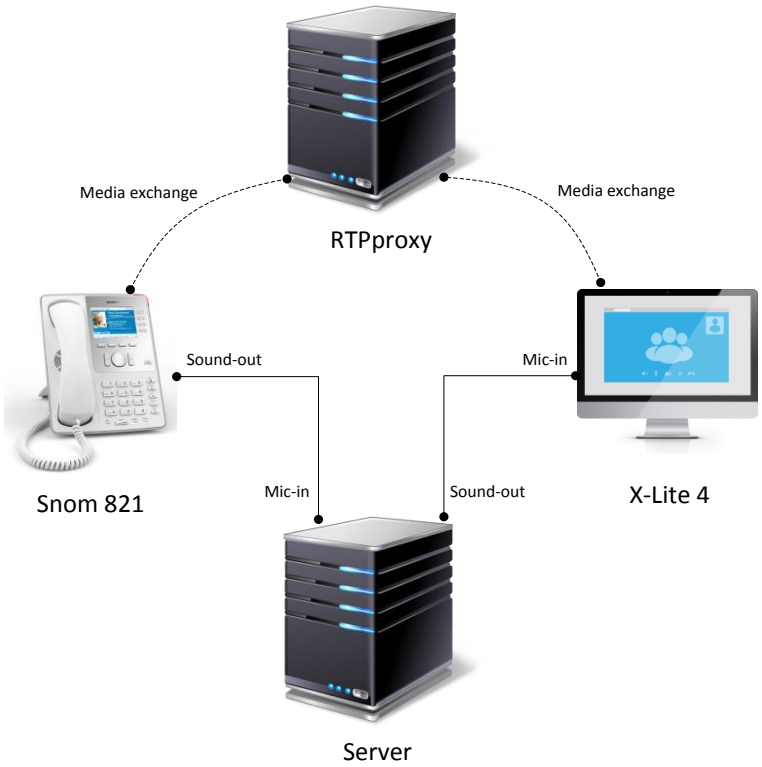

Figure 70 - RTPproxy test configuration

Figure 71 represents the captured media, with the original audio file being again on top and the recording below.

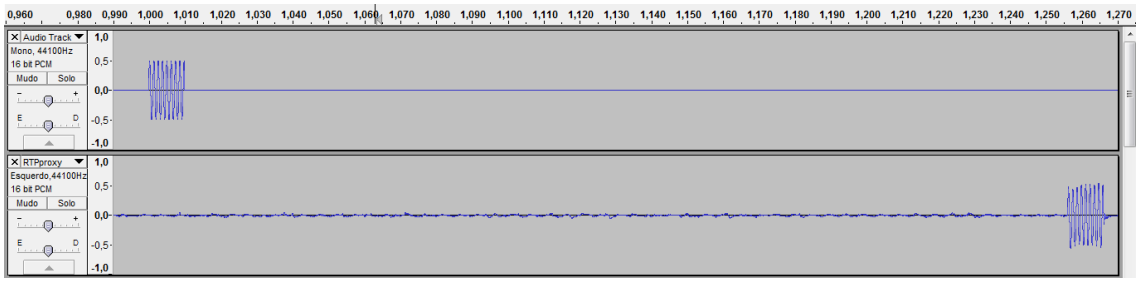

Figure 71 - RTPproxy audio comparison

As in the scenario above, the first audio spike featured in the original track starts at about the 1 second mark but, in this case, the first spike in the recording appears at approximately 1.255 seconds. A difference of 255 ms, an added 5 ms when comparing to the previous test result, this is a possible consequence from the media bridging established or the proxy's own implementation.

5.3.3. **AS call**

In this test, a call was triggered by the API to both parties, with connections being established between the clients and the AS, where media is exchanged during the call session – i.e., audio relay is made in the AS. The results achieved in this case can be seen in Figure 72.

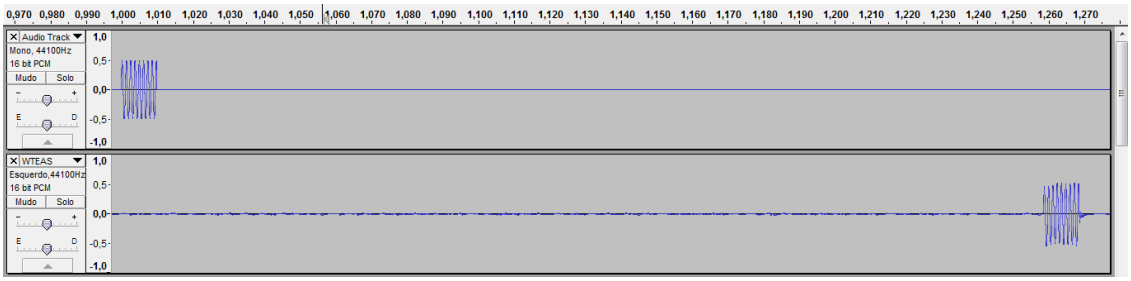

Figure 72 - AS audio comparison

Here, the first spike in the call recording appears at 1.26 seconds, for a total difference of 260 ms (considering the original track). As one can see, the AS behaves well in this realm, with its performance comparable to the RTPproxy's.

5.3.4. **WebRTC calls**

The first part of this test is composed of an end-to-end scenario where a call is established between the WWC client and the Snom phone, exposing the gateway's influence amidst the media path. The results can be seen in Figure 73.

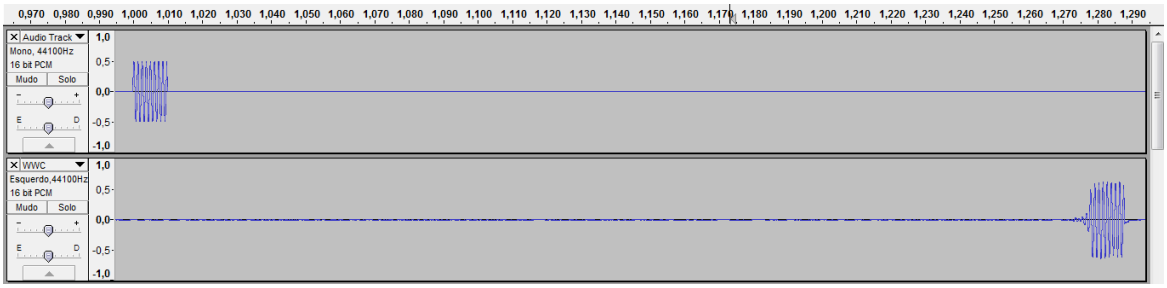

Figure 73 - WWC audio comparison

As one can see, the first spike now starts at around 1.28 seconds, 20 ms's more than in the previous test case or a total difference of 280 ms from the original audio track. The second part of this test features a scenario that involves not only the WWC client, but the AS as well, with Figure 74 representing its test configuration.

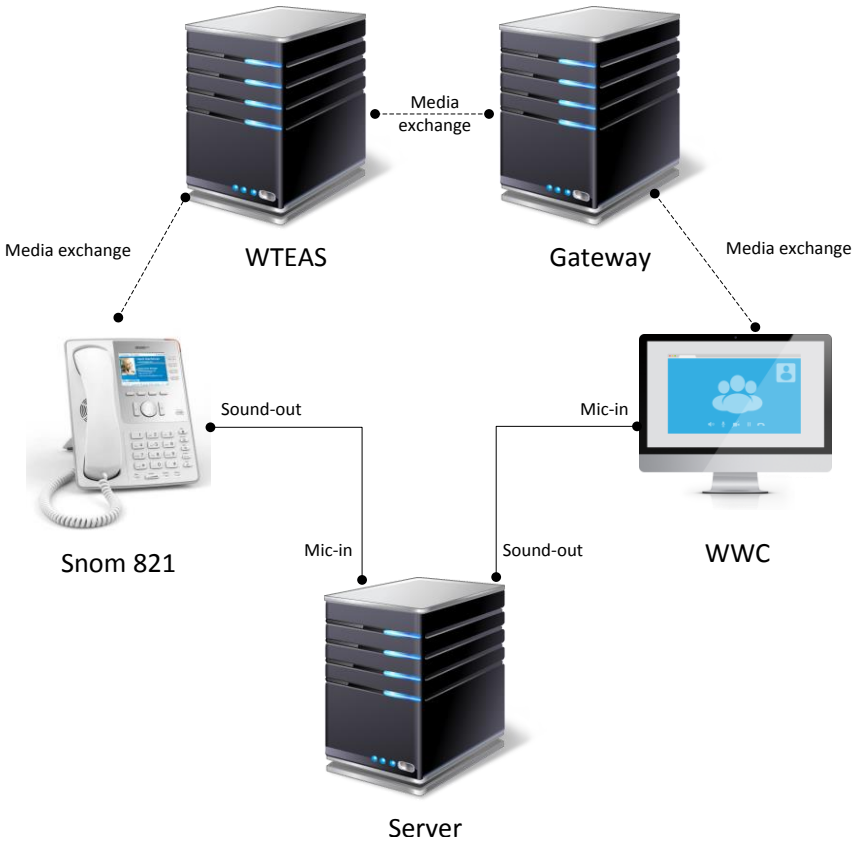

Figure 74 - AS/WWC test configuration

When comparing this previous test result to this test, where the AS is involved for media relay, one can see that the first spike (in Figure 75) is triggered at around 1.29 seconds, meaning that the AS added a 10 ms delay, minimally impacting the solutions performance in terms of audio transmission.

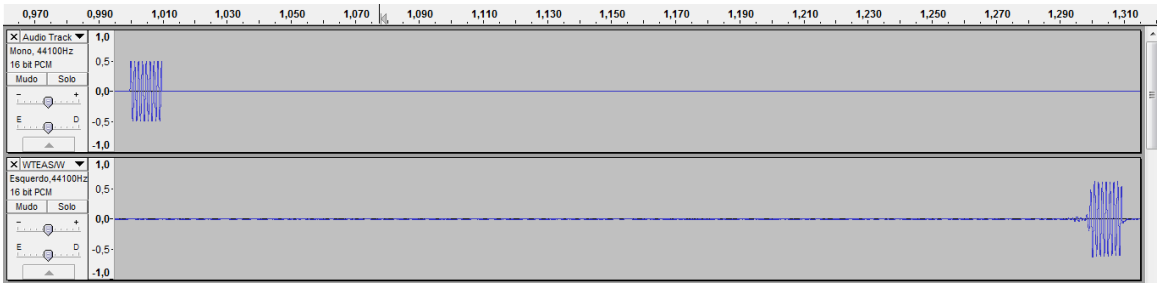

Figure 75 - WWC/AS audio comparison

At an average delay of 290 ms, performance is placed near ITU's "Some users dissatisfied" level, which has its lower-bound reference value set to 300 ms.

5.3.5. **GSM calls**

This test was done in order to establish a baseline comparison between the normal behavior for a traditional GSM (Global System for Mobile) phone call and a call made using the solution developed in this dissertation. The approach taken, depicted in Figure 76, was similar to the ones seen on the peer-to-peer test-case, with a sound-source playing the original audio, which is captured by the caller's speakers and sent to the callee. The transmitted audio is captured by the server's microphone, coming from the callee's loud-speakers. Again, no server component such as IMS core or WTE solution was used; this test-set was strictly made using smartphones registered to operator's networks. Although this difference resulted in a certain echo on the recording, along with some delay from the caller's speakers when acquiring the sound, this allowed for easy capture of the transmitted audio:

Figure 76 - GSM test configuration

The first test, depicted in Figure 77, was made between two different smartphones, functioning in the same operator and in GSM-mode. In this scenario, results showed that delay is situated near the 300 ms level (the difference between the start of the "signal generated" and the "signal sent"). As one can see by the difference between the start of the "signal sent" and the start of the "signal received", the device's speakers had an acquisition delay of about 110 ms.

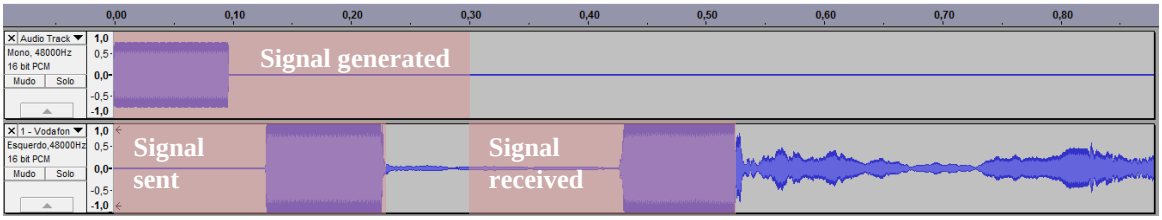

Figure 77 - GSM delay test result (same operator)

300 ms places the communication at the upper bound limit of ITU's "Some users dissatisfied" level of acceptance. Another test was done, although resulting in increasing delay times, with a call being made between different smartphones registered to different operators (Figure 78).

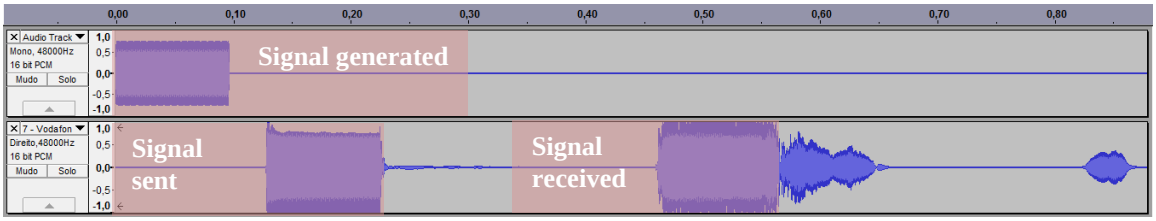

Figure 78 - GSM delay test result (different operators)

Results gathered indicate that delay was set at about 440 ms, going beyond ITU's upper bound limit on call delay recommendations (400 ms) and falling to ITU's "Many users dissatisfied" level of user acceptance.

Although a test with 4G support wasn't possible (fourth generation, referring to LTE-enabled smartphones), one was made with WCDMA (Wideband Code Division Multiple Access[87], a third generation technology allowing more speed in data exchange, among other improvements). This test can be seen depicted in Figure 79, with delay set at about 280 ms, placing at the bottom limit of ITU's "Users satisfied" level.

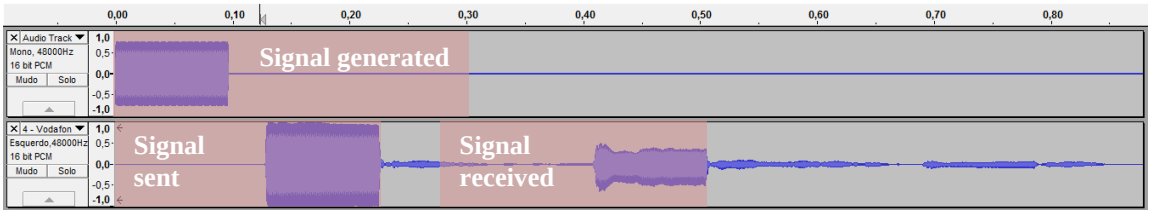

Figure 79 - WCDMA delay test result (same operator)

5.3.6. **Test results summary**

The most significant delay can be noticed when studying the gateway's impact, which is more likely to worsen call experience. The WWC, as explained previously, sends its media to the gateway, which only then forwards it to the necessary locations. This action greatly impacts the system, but is essential to maintaining control over the media flow and allow for transcoding mechanisms to be executed, if required.

The test results gathered are compiled in Table 3, expressed in milliseconds, with a column detailing delay difference from a test-case to the default one.

Test	Average delay	Delay difference
Peer-to-peer call	250	-
RTPproxy call	255	5
AS call	260	10
WebRTC call	280	30
WebRTC/AS call	290	40

Table 3 - Mouth-to-ear test results summary

One can see how comparing these results to the delay in a call between two mobile phones this is not fair, as network conditions for those calls were unknown, whereas in the solution's tests they were known and controlled, easing testing itself and creating more consistent results. However, they do provide a good reference to the delay that can be expected in this scenario and one can see how this solution's performance remains on par, since the lowest delays recorded for the calls were 280 ms (in WCDMA mode) and 300 ms (in GSM mode).

5.4. **Chapter summary**

Several conclusions may be extrapolated from the testing results gathered. Four main test-cases were executed, performance-wise: a scenario designed to test only IMS signalling throughput, another scenario including signalling and media exchange through an RTP proxy, one especially devised for the AS and another including WebRTC support. After a significant amount of testing and posterior analysis, one was able to find that above all, the reasons for the values fall on MTS performance. The tool remained a bottleneck over all the scenarios, not being able to cope with the load, signalling and media exchange necessities.

Focusing on the mouth-to-ear tests, one can see how these were affected by established baseline of a 250 ms delay, a test-case where only the IMS core was in use (and peer-to-peer media exchange was enforced). This result is considered substantially high, even when comparing it to the GSM/WCDMA tests made. If one was to look only at the added delay from the other components (AS and/or gateway), the results would be significantly lower - an average of 5 ms gained by the AS or 30 ms by the gateway. As such, an assumption is made on the solution's performance not becoming a bottleneck in other network configurations, as well as lower delays on the WebRTC-enabling components if conditions were bettered.

Overall, this testing phase was useful for gathering more in-depth information on the components behavior, enriching this solution. Moreover, even though the baselines values established were those gathered from the limited analysis, the belief on higher performance levels remains.

Chapter 6

Conclusion

Advances and research are being made to create a higher degree of interoperability between traditional telecommunications network and the web-centered world, with WebRTC motivating new use-cases around the IMS architecture. This dissertation is such an example, providing, at first, a background research on the tools available to create such a solution. Among these tools, one can find IMS cores and network components, as well as API designs that come to facilitate the modeling of the API exposing third party call controls. From these designs, the one chosen was the one delivered by GSMA and its OneAPI initiative. Influenced by major operators and influencing several projects around the globe, this initiative provided great groundwork over the way the API should be modeled, featuring specifications on different telephony aspects (from voice to messaging). The search for network components was also crucial, revealing solutions already used to provide WebRTC support over the IMS network and aiding in the search for an optimal solution – in this case, the usage of WIT Software's WebRTC-to-SIP gateway and client (WIT WebRTC Communicator). Taking into account the research and the knowledge gathered, it was possible to plan several use-cases and define platform requirements that reveal the solution's potential. Providing popular telephony functionalities, with the added value of the API designs in use, this solution also empowers users by enhancing service discovery, via a marketplace were applications are published.

The requirements defined came to affect the solution's roadmap and its development, where the implementation of certain features like call session managing, rich messaging exchanging and notification mechanisms was prioritized over others such as billing and chat functionalities. Re-using existing WIT projects and products, the solution created features an AS extended to expose OneAPI functions and a portal, custom-built to enable the management of every aspect of the platform. Additionally, WebRTC support was provided with the integration of a WebRTC gateway and client, the latter modified to accept messaging from the AS. The development effort of implementing and integrating the components meant that the requirements were almost entirely implemented. Moreover, throughout its development, additional changes to the requirements emerged, with the user provisioning mechanism designed (a custom API for network user registration in the platform) requiring the concept of "administrative applications", privileged applications that have exclusive access to that API. The decisions on re-using existing projects to fit this dissertation's needs enabled the exposing of a stable and feature-rich solution,

with its performance evaluated and proving to be on par with current expectations on mobile call delays, for example. Load-testing the platform revealed its behavior under stress, with the solution out-doing the tool in use across the different test-cases With the tool itself unable to handle all the call events created, the solution was able to perform efficiently in every test, never exposing bottlenecks that would impair communications or bring the system to a halt.

In short, the proposed objective of developing a proof-of-concept, featuring both telephony API and WebRTC support, is considered a success, with the platform succeeding in both features exposed and performance constraints. With the design decisions and components architecture, the platform has room to grow into a richer, enterprise-driven solution, able to be customized to fit any growing needs.

6.1. **Future work**

The current application server's implementation is functional and stable, providing already a high-level of error handling through a complex SIP layer and already packing some interesting features. Nonetheless, room for evolution exists across several domains.

The AS could eventually provide mechanisms for DTMF processing, chat messaging, API analytics and user handling. Rich messaging could also be enriched by supporting more content-types, increasing the amount of files that may be exchanged. Development could also be made on the SIP layer, with the server acting as a proxy for BYE requests, for example, empowering the user. Another layer that could be improved is the media one, providing support for more codecs in simultaneous, as well as a more complete announcement playback feature.

The portal has room to grow, from user experience to the administrator mechanisms. On its marketplace, features could be added to show more application details or even enable the rating of a published application. Development could be made on this area to reduce demands on administration of having to handle all requests.

Both API's could grow in terms of functionalities, focusing on other contexts such as billing or more complete user provisioning mechanisms. This would allow the creation of more enhanced use-cases, which would make the platform more flexible to other usage contexts.

In terms of the solution itself, further testing could be made to ensure its correct behaviour under stress and overall interoperability. With other performance tools and more powerful hardware, extended testing could be made to ensure the solution is able to handle higher numbers of concurrent calls and test its processing capabilities under intensive stress. Additionally, integration tests could be made, deploying the platform in a real operator network and testing its support for different clients and smartphones. Although federation between multiple operators was

not considered, this would be necessary for widespread deployment of the solution. In terms of the use-cases explored and implemented during the course of this dissertation's development, popular CRMs such as SugarCRM and Salesforce could be supported by creating plugins for these tools. This would properly show-off the solution's capabilities and the added value brought.

References

[1] K. Al-Begain *et al.*, *IMS: A Development and Deployment Perspective*, p.^pp. 316: John Wiley & Sons, 2009.

[2] P. Granstrom, L. Norell, and S. Akesson, *Converged service for fixed and mobile telephony*, Ericsson Review, 2009.

[3] *The Architecture and Benefits of IMS*, Dialogic Corporation, 2009.

[4] B. Williamson, "Over-The-Top (OTT): helping or hindering network investment?," *InterMEDIA*, vol. 41, no. 13, 2013.

[5] K. S. Jr, *Introduction to the IP Multimedia Subsystem (IMS): IMS Basic Concepts and Terminology*, F5 Networks, Inc., 2007.

[6] J. Rosenberg *et al.*, "RFC 3261 - SIP: Session Initiation Protocol," IETF, 2002.

[7] M. Handley, and V. Jacobson, "RFC 2327 - SDP: Session Description Protocol," IETF, 1998.

[8] Lynne, "CSCF in VoLTE – The P-CSCF (Part 1 of 4)," Award Solutions, Inc., 2013.

[9] *Introduction to IMS*, Ericsson AB, 2007.

[10] "Universal Mobile Telecommunications System (UMTS); Policy and charging control over Gx reference point (3GPP TS 29.212 version 7.4.0 Release 7)," *Gx Reference Point*, ETSI, 2008.

[11] M. Poikselkä *et al.*, *Voice over LTE (VoLTE)*: John Wiley & Sons, 2012.

[12] *Voice and Video calling over LTE*, Ericsson, 2012.

[13] *IMS procedures and protocols*, Spirent, 2014.

[14] M. Networks. "About Clearwater," http://www.projectclearwater.org/about-clearwater/, 2013.

[15] M. Networks. "Clearwater Architecture," http://www.projectclearwater.org/technical/clearwater-architecture/, 2013.

[16] V. Technologies. "SIP Call Transferring," http://www.vocal.com/sip-1/call-transferring/, 3014.

[17] F. FOKUS. "OSIMS - The FOKUS Open Source IMS Core," http://www.fokus.fraunhofer.de/en/fokus_testbeds/open_ims_playground/components/osims/index.html, 2014.

[18] Kamailio. "Kamailio SIP Server," http://www.kamailio.org/w/, 2014.

[19] Kamailio. "What is the IP Multimedia Subsystem (IMS) in Kamailio 4.0?," http://www.kamailio.org/w/2013/05/ims-kamailio/, 2013.

[20] Kamailio. "FreeSwitch as Media Server and SBC for Kamailio 3.1," http://www.kamailio.org/w/2010/11/freeswitch-as-media-server-and-sbc-for-kamailio-3-1/, 2014.

[21] R. Manson, *Getting Started with WebRTC*, p.^pp. 114: Packt Publishing, 2013.

[22] M. Baugher *et al.*, "RFC 3711 - The Secure Real-time Transport Protocol (SRTP)," IETF, 2004.

[23] E. R. Stewart, "RFC 4960 - Stream Control Transport Protocol," IETF, 2007.

[24] I. Baz Castillo *et al.*, "The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)," Internet Engineering Task Force, 2014.

[25] S. Wilkins, *Basic NAT Concepts and Configuration*, Cisco Press, 2011.

[26] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," Internet Engineering Task Force, 2010.

[27] J. Rosenberg *et al.*, "Session Traversal Utilities for NAT (STUN)," Network Working Group, 2008.

[28] Anymeeting. "Anymeeting - Features," http://www.anymeeting.com/Free-Web-Conferencing-Features.aspx, 2014.

[29] &yet. "Talky," https://talky.io/.

[30] LiveCom. "webRTC:bring us closer," http://lwork.hk:8086/###, 2014.

[31] I. Microsoft Open Technologies. "HTML5Labs - FAQ," http://html5labs.interopbridges.com/prototypes/cu-rtc-web/cu-rtc-web/faq, 2014.

[32] J. Paoli. "Customizable, Ubiquitous Real Time Communication over the Web (CU-RTC-Web)," http://blogs.msdn.com/b/interoperability/archive/2012/07/28/customizable-ubiquitous-real-time-communication-over-the-web-cu-rtc-web.aspx, 2013.

[33] GTalk-To-VoIP. "Flash-to-VoIP," https://www.flash2voip.com/.

[34] D. Telecom. "flash2ims," https://code.google.com/p/flash2ims/.

[35] D. Bubley. "WebRTC forecasts upgraded: mobile support accelerating," http://webrtcstats.com/webrtc-forecasts-upgraded-mobile-support-accelerating/, 2014.

[36] WebRTCstats. "First WebRTC statistics in the World!," http://webrtcstats.com/first-webrtc-statistics/, 2013.

[37] P. S. Inc. "EasyRTC Enterprise," http://www.easyrtc.com/, 2014.

[38] E. Reeves. "5 Facts About Web Real Time Communication (WebRTC)," http://blogs.ixiacom.com/ixia-blog/5-facts-about-web-real-time-communication-webrtc/, 2014.

[39] J. Bankoski *et al.*, "VP8 Data Format and Decoding Guide," IETF, 2011.

[40] L. Miniero. "Are all WebRTC gateway the same?," 2014; http://lphs.acmepacket.com/blog/bid/175527/Are-all-WebRTC-gateways-the-same.

[41] *Underdstanding Session Border Controllers*, FRAFOS, 2013.

[42] *Security Gateways Versus Session Border Controllers for Protecting Fixed-Mobile Convergence Networks*, Reef Point Systems, 2006.

[43] E. Cheung, "Implementing Endpoint Services Using the SIP Servlet Standard," in International Conference on Networking and Services, 2009, pp. 6.

[44] L. Red Hat Middleware. "Mobicents SIP Servlets," http://www.mobicents.org/products_sip_servlets.html, 2014.

[45] P. Sailfin. "Sailfin Project," https://sailfin.java.net/, 2013.

[46] Oracle, "Oracle Communications Converged Application Server ", 2010.

[47] Oracle, "Oracle Communications Session Border Controller ", 2013.

[48] Oracle, "Oracle Communications WebRTC Session Controller Solution," 2013.

[49] D. Telecom. "webrtc2sip," http://webrtc2sip.org/, 2013.

[50] D. Telecom. "webrtc2sip - Smart SIP and Media Gateway," https://code.google.com/p/webrtc2sip/, 2013.

[51] D. Telecom. "sipml5 - World's first HTML5 SIP client," http://sipml5.org/, 2013.

[52] R. Lidstone. "Enterprise, SMB WebRTC Adoption on the Rise," http://www.webrtcworld.com/topics/webrtc-world/articles/351721-enterprise-smb-webrtc-adoption-the-rise.htm, 2014.

[53] P. Inc. "Plivo Docs - APIs for Voice, SMS and Text Messaging," http://plivo.com/docs/api/, 2014.

[54] P. Inc. "Plivo Launches SDK That Lets Developers Connect WebRTC And SIP," http://plivo.com/blog/plivo-launches-sdk-that-lets-developers-connect-webrtc-and-sip/, 2014.

[55] P. Inc. "Plivo Framework," https://github.com/plivo/plivoframework/, 2014.

[56] "GSMA One API Exchange Architecture for cross-operator network APIs," GSMA, 2013, p. 6.

[57] "Open Service Access (OSA); Parlay X Web Services; Part 2: Third Party Call (Parlay X 2)," ETSI, 2008.

[58] Pragmateek. "JSON vs. XML: Some hard numbers about verbosity," http://www.codeproject.com/Articles/604720/JSON-vs-XML-Some-hard-numbers-about-verbosity, 2014.

[59] K. Mikoluk. "JSON vs XML: How JSON Is Superior To XML," https://www.udemy.com/blog/json-vs-xml/, 2013.

[60] "OneAPI V2.0 Voice Call Control Rest," Aepona, 2012.

[61] "OneAPI SMS REST V2.0," Aepona, 2012.

[62] "OneAPI Multimedia Messaging REST V2.0," Aepona, 2012.

[63] J. Rosenberg *et al.*, "Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)," IETF, 2004.

[64] P. Dix, *Service-Oriented Design with Ruby and Rails*, p.^pp. 320: Addison-Wesley Professional, 2010.

[65] Apigee. "Apigee Developer Platform," https://developers.apigee.com/app-services, 2014.

[66] E. E. Burger, J. Van Dyke, and A. Spitzer, "Basic Network Media Services with SIP," IETF, 2005.

[67] T. Erl, *SOA: Principles of Service Design*: Prentice Hall PTR, 2007.

[68] A. Handa, *System Engineering for IMS networks*: Newnes, 2009.

[69] C. Nedelcu, *Nginx HTTP Server*, p.^pp. 318: Packt Publishing, 2013.

[70] M. Kulkarni, and Y. Cosmadopoulos, *SIP Servlet Specification, version 1.1*, 2008.

[71] O. Corporation. "Jersey - RESTful Web Services in Java," https://jersey.java.net/, 2014.

[72] T. N. Project. "Netty Project," http://netty.io/, 2014.

[73] E. A. Johnston *et al.*, "Session Initiation Protocol Service Examples," IETF, 2008.

[74] M. Poikselkä, and G. Mayer, *The IMS: IP Multimedia Concepts and Services*, 3rd Edition ed.: Wiley, 2009.

[75] J. Rosenberg *et al.*, "Session Initiation Protocol (SIP) Extension for Instant Messaging," IETF, 2002.

[76] "RCS-e - Advanced Communications: Services and Client Specifications," GSMA, 2012.

[77] MTS. "MTS," http://mts.arm-tool.com/, 2014.

[78] R. Gayraud. "Welcome to SIPp," http://sipp.sourceforge.net/, 2014.

[79] P. Zadrozny, P. Aston, and T. Osborne, *J2EE Performance Testing with BEA WebLogic Server*: Apress L.P, 2003.

[80] T. Senart. "tsenart/vegeta - HTTP load testing tool and library," https://github.com/tsenart/vegeta, 2014.

[81] T. Audacity Team. "Audacity: Free Audio Editor and Recorder," http://audacity.sourceforge.net/, 2013.

[82] MTS, "Documentation MTS - User Manuals," 2014.

[83] I. Sippy Software. "Sippy SIP B2BUA," http://www.b2bua.org/, 2014.

[84] Google. "The Chromium Projects - Audio/Video," http://www.chromium.org/audio-video, 2014.

[85] C. Agastya, D. Mechanic, and N. Kothari, “Mouth-To-Ear Latency in Popular VoIP Clients,” Columbia University, New York, 2009.

[86] "Series G: Transmission Systems and Media, Digital Systems and Networks," ITU, 2003.

[87] R. Tanner, and J. Woodard, *WCDMA: Requirements and Practical Design*: John Wiley & Sons, 2005.

[88] R. Fielding *et al.*, "RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1," IETF, 1999.

[89] V. Consortium, "vCard - The Electronic Business Card," 1996.

[90] J. Rosenberg, "RFC 3680: A Session Initiation Protocol (SIP) Event Package for Registrations," IETF, 2004.

A) Requirements

In this annex, a high-level view of the requirements for the system were given, discarding low-level details like internal workflows of the stated operations - how a RTP channel will be opened to provide media exchange between a third-party application and a SIP endpoint already registered on the network, for example. Although some features were discusses, not all will be integrated into the solution at this time. For instance, in order to create an interesting proof-of-concept, features such as call controls were prioritized over those of presence services.

A.1) Functional requirements

Functional requirements represent what the solution needs to do and what is available for interaction, from a user or developer perspective. For example, being able to add an application to the portal or triggering a call via the API are functional requirements, whereas security and performance are general requirements – capabilities that need to exist and are not directly used by end-users.

A.2.1) API/WebRTC requirements

Over the course of the market analysis, with the research and analysis made on already available telephony API's, certain functionalities were seen more often exposed than others. An example of this is call control versus DTMF processing (Dual-Tone Multi-Frequency, referring to the ability to send phone digits over a call). While both API's support call placement, only Plivo processed DTMF and offered mechanisms for developers to capture and answer it (as one would to in a call-service, for example, dialing the numbers necessary to execute the desired operations). Furthermore, these APIs already offer support for WebRTC client applications, enabling the developers to deploy an application using this technology and, for instance, place calls to PSTN numbers. Such features are the main objective of this dissertation and will be addressed momentarily.

Authentication

A mandatory step in user registration and handling, authentication methods must be present in order to address several more sensitive needs, requirements presented in Table 4 - Table 6:

- Application authentication, required to view application details (such as analytics);
- Developer authentication, allowing this person to access privileged account settings and view account details;
- User registration and status announcement within the network, allowing him to interact with other users already registered and available.

Requirement ID	Function	Description
MRS_API_AUTH_001	**Application authentication**	An application will have to access authentication mechanisms and receive authorization in order to receive access to networks resources.
MRS_API_AUTH_002	**Get application details**	A developer should be able to access application statistics via the API, accessing usage metrics (number of users currently logged-in or number of calls being made concurrently).

Table 4 - Application authentication functions

Requirement ID	Function	Description
MRS_API_AUTH_003	**Developer login**	A developer may login to his account at any time, in order to use all
MRS_API_AUTH_004	**Get developer details**	A developer should be able to get his account details such as his credentials or checking his status (if authorized for operations, for example).

Table 5 - Developer authentication functions

Requirement ID	Function	Description
MRS_API_AUTH_005	**User login**	With this function, a user will be able to re-register in the network, enabling interaction with other users.
MRS_API_AUTH_006	**User sign-up**	Users can sign-up to use a published application's services.

Table 6 - User authentication functions

Call operations

Whereas with a SIP endpoint one can use the IMS network normally, sending messages to other registered users or placing calls without necessarily accessing the API, WebRTC callers won't be able to do the same without proper support – these users wouldn't have the mechanisms for session signalling to connect to other users. Providing these signalling functionalities and call operations (presented in Table 7) for third-party applications will be the API's responsibility.

Requirement ID	Function	Description
MRS_API_CALL_001	**Place call**	Call may be started from the API, signalling the network and sending the request to a SIP endpoint, for example, from a WebRTC client in order to initiate a call between both.
MRS_API_CALL_002	**Answer call**	A call may be answered by a user upon network notification.
MRS_API_CALL_003	**Hang-up call**	In the end of a call session, a user may decide to hang-up his call, terminating the session.
MRS_API_CALL_004	**Reject call**	A callee may reject an incoming call immediately after being notified.
MRS_API_CALL_005	**Hold call**	A call may be put on hold by any of the call participating parties, notifying the different users of such an event (something that will be handled by a different function) and stopping the conversation momentarily.
MRS_API_CALL_006	**Resume call**	After putting a call on hold, the user that made such a decision may resume the call.
MRS_API_CALL_007	**Deflect call**	A user may be able to deflect a call to another party upon notification of an incoming call.
MRS_API_CALL_008	**Transfer call**	While participating on an on-going call session, such call may be transferred by the callee to another party (either after warning the party or not, which constitutes an unattended/attended call transfer, respectively).
MRS_API_CALL_009	**Add to call**	A user may be added to an on-going call session between other users.
MRS_API_CALL_010	**Invite to call**	A user may be invited to participate in an on-going call session between other users.
MRS_API_CALL_011	**Play announcements within a call**	Announcements may be played in a call, an event seen normally when on hold, for example. Additional features such as the uploading of a song to play during the announcement are also considered.

MRS_API_CALL_012	**Send DTMF**	Discussed previously, the API should provide DTMF sending, in order to interact with IVR (call robots), for instance.
MRS_API_CALL_013	**Receive DTMF**	With this function, DTMF processing would be available for developers to create richer applications.
MRS_API_CALL_014	**Click-to-Call**	Click-to-Call could be made available so developers could provide C2C placement mechanisms – alike Big Blue Button mechanisms.
MRS_API_CALL_015	**Click-to-Dial**	Click-to-Dial, which defers from the previous in its flow, could be made available to developers.

Table 7 - Call operations overview

Messaging

This messaging feature-set in intended to enable users to send short/multimedia messages or create instant messaging sessions with other users. These requirements can be seen in Table 8.

Requirement ID	Function	Description
MRS_API_MESSAGE_001	**Send message**	The user must be able to send text or multimedia messages to other users. Although the resulting flows for a text or multimedia message are different, this distinction should be hidden on the API, qualifying both simply as "messages" and letting internal mechanisms process them in their own proper manners.
MRS_API_MESSAGE_002	**Receive message**	A receiver must be opened to message receiving in order to fully interact with users wishing to communicate with him.
MRS_API_MESSAGE_003	**Create multimedia session**	With the possibility of creating a multimedia session, a user would be able to communicate via instant messaging with other users, sending text or other multimedia contents.
MRS_API_MESSAGE_004	**Invite users to multimedia session**	A user may be invited to participate in an on-going session.

Table 8 - Messaging functions

Notifications

Users need to be notified of API/gateway communications to be able to address such events properly and engage other users in the interactions. Without these, normal operations wouldn't be possible – users wouldn't receive any feedback on call processing, for example. Table 9 represents these requirements, along a small description of each.

Requirement ID	Function	Description
MRS_API_NOTIFICATION_001	**Incoming communication**	Notification of incoming calls, messages or others must be forwarded to the expected user, warning him of the upcoming event.
MRS_API_NOTIFICATION_002	**Call state**	Notifications must be implemented to give a caller/callee the knowledge if his call is on hold, for example.
MRS_API_NOTIFICATION_003	**Message delivery**	Message delivery states should be returned to the sender, enabling him to assess whether his message has been delivered, is pending or has been rejected
MRS_API_NOTIFICATION_004	**Authentication signalling**	Signalling must be made available for all users, in order to control their sessions status (if they are authenticated/registered in the network, for example)

Table 9 - Notification functions

These notifications will be received, by the user, using the channels he has opened upon registration. These, which are already used for signalling the network for call session establishment, for example, will also be used by the network to signal the application of incoming calls or messages, call states, among others.

Presence

With the functionalities in Table 10, users will have access to presence services, able to announce their status to other users and create "Friend" lists (subscriptions to presence lists).

Requirement ID	Function	Description
MRS_API_PRESENCE_001	**Publish presence status**	A user may publish its status, informing the network of such modification.

MRS_API_PRESENCE_002	**Subscribe to updates**	A user may subscribe to receive another user's status changes. Examples of this are popular chat services, where adding a user will automatically subscribe to his notifications.
MRS_API_PRESENCE_003	**Unsubscribe to updates**	Just like subscribing, a user may unsubscribe, stopping the receiving of notifications from other users updates.

Table 10 - Presence functions

A.2.2) Portal requirements

As done with the API, several competitors' dashboards/portals where analyzed in order to better design this dissertation's portal. With certain functionalities that stood out, such as usage analytics, logging access and billing summary, several conclusions were draw regarding what should be included. This will be a relevant part of the solution: designing a portal with features similar to solutions already available will help developers that have used such services to better acquaint themselves with the solution here presented. The portal is divided into different sections, corresponding to the role of the system's users (administrators, developers and end-users), the reporting functionalities, documentation and application catalog, which will be analyzed in the next topics.

Reporting - Analytics/Logging

With the functionalities in Table 11, users are able to view information regarding service usage and the latest activities they have conducted in the network, for example.

Requirement ID	Function	Description
MRS_PORTAL_ANALYTICS_001	**API usage**	With this functionality, metrics regarding most used operations, for example, may be reviewed.
MRS_PORTAL_ANALYTICS_002	**Application usage**	Using this function, an application usage metrics may be accessed.
MRS_PORTAL_ANALYTICS_003	**Activity logs**	Latest actions taken in the system may be overviewed with this functionality, a functionality used by administration for monitoring purposes

Table 11 - Available analytics functions

In this last feature, seen in Table 12, not only are these logs useful for monitoring purposes, but could be adapted for users, enabling them to view their latest activity (call and messaging registry or a time log of their logins). Along with recent calls, billing features can also be implemented, enabling a developer/user to keep financial track of his usage of the network.

Requirement ID	Function	Description
MRS_PORTAL_BILLING_001	**Usage analytics**	With this functionality, billing summaries may be accessed in order to control operational costs.

Table 12 - Billing function

A summary could be shown, showing metrics of calls made or messages sent in line with recent activities, showing the costs of the system's usage. Although not completely focused on at this time in the dissertation, one can envision billing related functionalities that the portal would provide, such as the creation of CDR's (Call Detail Record) for later analysis.

Administration

Administration will be responsible for controlling the publishing of applications and platform users. Table 13 depicts the requirements for the administrative actions contemplated.

Requirement ID	Function	Description
MRS_PORTAL_ADMIN_001	**Application approval**	Administration is given the responsibility of approving application before these can be published and used by other users.
MRS_PORTAL_ADMIN_002	**Disable application**	An application may be disabled in case of miss-conduct, such as network abuse.
MRS_PORTAL_ADMIN_003	**Developer approval**	Developers may be approved or refused to operate under certain conditions, something that can be controlled using this function.
MRS_PORTAL_ADMIN_004	**Delete developers**	Developers may be banned from using the service, for miss-conduct for example, by the administration.
MRS_PORTAL_ADMIN_005	**Notify pending requests**	Administration should be notified of pending approvals requests made by developers

Table 13 - Administrative actions

Application catalog (Marketplace)

The catalog or marketplace is where any user will be able to view all published applications and their details, a feature commonly seen in mobile environments (and depicted in Table 14).

Requirement ID	Function	Description
MRS_PORTAL_MARKET_001	**List applications**	A full application listing will be presented, enabling overview of available services.
MRS_PORTAL_MARKET_002	**Show application description**	As an application is selected, an overview of its details may be accessed, presenting useful information regarding the services it provides.

Table 14 - Marketplace functions

Application management

With the features in Table 15, application management is assured, with the administration able to approve and authorize an application usage of the network and with the developers provided of a way to add their applications for approval. Quality assurance features are required, with the administration also able to block/disable an application of using the network.

Requirement ID	Function	Description
MRS_PORTAL_DEV_001	**Send application for approval**	An application may be sent in for approval, being added to the network if accepted.
MRS_PORTAL_DEV_002	**Check application status**	After sending the application for approval, the developer may check its status (if pending or approved/denied)
MRS_PORTAL_DEV_003	**Add a developer to an application**	A developer may be added to an application through this function.

Table 15 - Application management operations

The publishing flow would follow a path similar to other application markets: a developer would add an application to the portal, pending its approval by the administration. After conducting a review of the application, the administration would be able to approve it, resulting in its publishing in the marketplace and authorizing its usage of the network's resources. At this point, a user would simple login, search the marketplace for the application and use immediately.

Developer management

Containing developer-oriented features, the requirements in Table 16 will allow developers to manage their accounts, modifying or accessing personal information.

Requirement ID	Function	Description
MRS_PORTAL_DEV_004	**Create account**	A developer may create his account at any time, pending approval by administration.
MRS_PORTAL_DEV_005	**Edit account**	Personal details may be edited by an individual.
MRS_PORTAL_DEV_006	**Delete account**	A developer may delete his account at any time.

Table 16 - Developer management operations

User

The end-user of the system, this is the entity that ultimately uses the applications that are published and indeed the network for communication purposes. A user must be provisioned before using the services, in the network and the applications, ensuring its access to both. Although its registry and deletion are an operator's responsibility, the user may modify personal information or configure/block the services he uses (Table 17).

Requirement ID	Function	Description
MRS_PORTAL_USER_001	**Edit account**	Personal details may be edited.
MRS_PORTAL_USER_002	**Block application**	An application may be blocked by a user, deleting its data from the system.

Table 17 - User actions

Not just to ensure the access to all the services, a user is limited in the concurrent actions he can take on the system, a security feature (presented in Table 18) that must will prevent abuse or attacks on the network.

Requirement ID	Function	Description
MRS_PORTAL_USER_003	**User provisioning**	User provisioning will be available through the portal, providing support for user creation and network registration.
MRS_PORTAL_USER_004	**Application/Service provisioning per user**	In order to protect the solution from inappropriate use, limits are imposed as to how many calls to the API a single user can make at any given time.
MRS_PORTAL_USER_005	**Device provisioning per user**	Multiple devices may be related to a single user, using IMS's mechanisms for device handling.

Table 18 - User management operations

A.3) General requirements

General requirements are those that are not available for user interaction, but constraints to the solution, features that it should strive for. Table 19 defines the identifications for the non-functional requirements.

Requirement ID	Function	Description
MRS_SYSTEM_API_001	**Provide an abstraction layer for network features**	The system must provide an abstraction layer for underlying network features such as call control, signalling, among others.
MRS_SYSTEM_PORTAL_001	**Provide a backoffice for all users**	A portal must be made available to act as a back-office for administration, users and developers. Using this portal, all users can access personal information and use the system.
MRS_INTEROP_WEBRTC_001	**Support for WebRTC**	WebRTC support must be made available by the solution.
MRS_INTEROP_NETWORK_001	**Support for calls between networks**	Calls between the home and external networks must be ensured.
MRS_CONC_MULTIUSER_001	**Support for multiple users**	WTE must support the processing of concurrent actions from multiple users, at any given time.

MRS_PERFORM_SCALE_001	**Provide scalability mechanisms**	The solution must scale in accordance to increasing load and requests.
MRS_PERFORM_AVAIL_001	**Provide availability measures**	The solution must have availability measures, minimazing possible down-time.
MRS_SEC_PROVISION_001	**Provisioning methods**	Users, developers and applications must be provisioned by administration in other to use all desired services.
MRS_SEC_PRIVACY_001	**Privacy insurance**	All communications must be secured and provide encryption support.

Table 19 - General requirements

B) OneAPI layer

This API, modeled after the OneAPI specifications, is exposed by the AS, in order to bring third party call control capabilities to authorized third party applications. This is one of the main focuses in this dissertation, with the OneAPI integration being considered as a feature that truly adds value to the solution. Throughout this annex, certain changes to the original OneAPI specification will be analyzed, alongside proper explaining as to why this was done and the impact it might have to a third-party developer that has previously come across such design. All other details are taken "as is" and implemented according to OneAPI's recommendations on Voice (OneAPI Voice Call Control[60]), SMS (OneAPI SMS[61]) and MMS (OneAPI MMS[62]).

B.1) Authentication deviation

The API authenticates the requests it receives by looking for an application token parameter in the resource called. The logic behind this decision is a design one: all applications that interact with the solution are seen as third party and will need to go through an authorization process, where the database is queried for whether the application is authorized to make requests or not. Unfortunately, the OneAPI specification does not contemplate this token-based authentication, relying on certificate exchanging and HTTP-based authentication mechanisms.

Certain alternatives exist, such as integrating the application token in the request bodies, but this had two major disadvantages: firstly, it was a greater deviation from the original specification, altering the bodies sent, and secondly, this meant that in a terminate call request, which is expected to be accessed via a DELETE method, would also need a JSON body with the token. Although the first disadvantage is a trade-off between needing to bring the desired features to the solution and maintaining OneAPI specification, the second is a significant obstacle since no expected body is present in a DELETE method[88]. By using such approach, the result would be a defective API design which would result in several obstacles for developers using the solution, such as the possibility of breaking client implementation.

Another alternative could be the sending of the applicationToken via an URL encoded parameter. However, this brings its own obstacles: in order to maintain pattern across the API design, all other methods would have to contain this URL encoded parameter and, in POST methods, would also have to bring JSON bodies. This would bring a conflict in content types accepted by the API (as OneAPI states that only JSON content types should be exchanged) and would possibly be an obstacle to HTTP client implementations.

As such, in order to minimize impact and deviations to the OneAPI integration and maintain a sensible pattern across the API methods, the application token is sent in the resource URL as show in Snippet 4.

```
http://localhost:9090/WTE/resources/2013-12-
12/<applicationToken>/thirdpartycall/callSessions/<callSessionId>
```

Snippet 4 - Example resource showing the application token location

This is not a perfect solution as it still breaks specifications but is seen as a less intrusive way of sending the token, as developers using this solution would only need to add a variable to the path base of their OneAPI implementations and use the solution, requiring less development effort.

B.2) Features

As referred above, this API will be used to provide all the call control, notification and authorization needs with which to use on developed applications.

B.2.1) User authentication

This feature-set refers to the authenticating methods provided by the API. The solution already authorizes the request based on the token given, as referred before, but a different method is used to authenticate network users that log in the applications. In order to enable a third party application to authenticate network users, the AS will expose a simple user authentication interface. This is a feature that is not planned in the OneAPI specification and, as such, is exposed in a URL base different from the rest of the third party call control methods.

Request

To authenticate a user, a developer will have to make an HTTP GET request, with a body such as the one shown in Snippet 5, URL featuring the application token and content-type set to application/json (Snippet 6).

```
    { "user" : { "email" : “user@email.here”,   "password" :
“userPassword” } }
```

Snippet 5 – Example user authentication request body

```
curl -v -H "Content-Type:application/json" -X GET
http://localhost:9090/WTE/resources/2013-12-
12/<applicationToken>/userAuthentication -d
'{"user":{"email":"bcruz@mail.pt", "password":"test"}}'
```

Snippet 6 – Example user authentication request

Response

The above request will wield a response such as the one in Snippet 7.

```
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 05 Mar 2014 12:35:41 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"user":{"credentials":true}}
```

Snippet 7 - Example user authentication response

The result of "credentials" field may be true or false, depending whether the credentials that were sent in the request are valid or invalid.

B.2.2) Sending an SMS

With this feature, the developer is able to send a textual message to a desired user, a pager-mode message following OneAPI's SMS specification[61].

Request

Defined in Snippet 8 is the JSON body required for this request, placed to the API via the POST method.

```
{ "outboundSMSMessageRequest":
  { "address": [ "+351910000001" , "+351910000002" ],
    "clientCorrelator": "67891",
    "outboundSMSTextMessage": {
       "message": "Test Message" },
    "receiptRequest":{
    "notifyURL":"http://www.yoururl.here/notifications"
    }, "senderAddress": "351910000003" } }
```

Snippet 8 - Example SMS send request body

An example curl request can be seen in Snippet 9, again featuring application token and body.

```
curl -X POST -H "Content-Type:application/json"
http://localhost:9090/WTE/resources/2013-12-
12/<applicationToken>/smsmessaging/outbound/+351910000003/requests
-d '{"outboundSMSMessageRequest": {"address":
["+351910000001","+351910000002"], "clientCorrelator":
"67891","outboundSMSTextMessage": {"message": "Test Message"},
"receiptRequest":{"notifyURL":"http://www.yoururl.here/notificatio
ns"},"senderAddress": "+351910000003"}}'
```

Snippet 9 - Example SMS send request

Response

A request such as Snippet 9 would result in a response similar to Snippet 10.

```
< HTTP/1.1 201 Created
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Mon, 10 Mar 2014 11:20:48 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"resourceReference": {"resourceURL":""}}
```

Snippet 10 - Example SMS send response

At this point, the solution responds with the “resourceURL” parameter in blank, since no mechanism for message delivery state was implemented.

Messaging Flow

This API request/response reflects the interaction in Figure 80 with the network components:

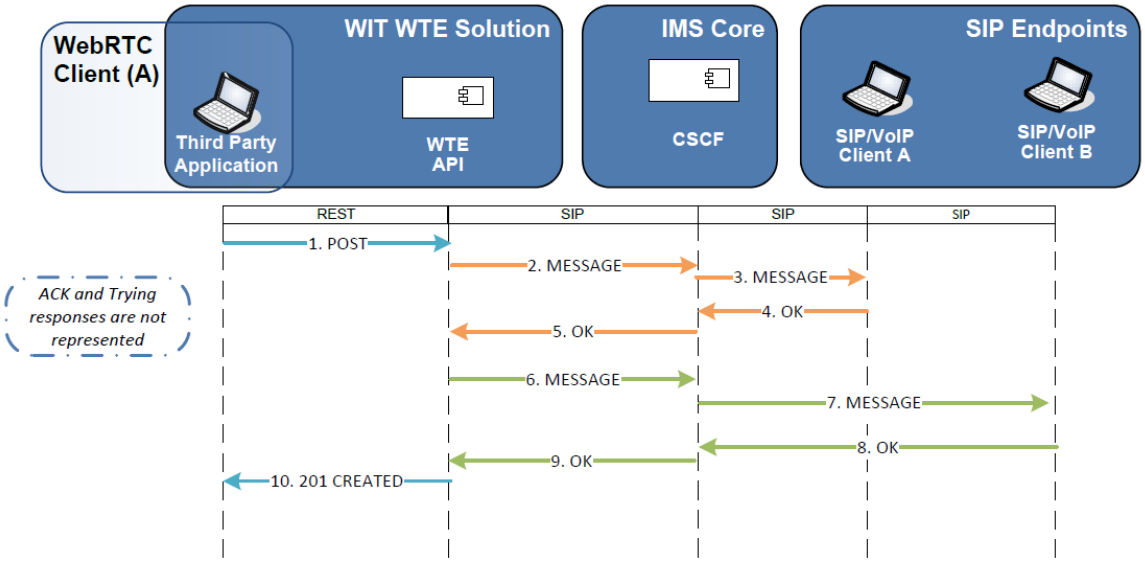

Figure 80 - Messaging flow

B.2.3) Sending an MMS

With these functionality, a message with multimedia content can be sent to users (only supported by the WebRTC client, however, due to compatibility issues between devices). This feature does not implement the OneAPI specification on MMS’s[62] entirely, being the deviation discussed below.

Request

This feature is available by placing a multipart POST request to the solution, with the resource featuring both application token and sender address. This multipart request can be seen in Snippet 11 and Snippet 12.

```
------------------------------a83abfca62f9
Content-Disposition: form-data; name="outboundMessageRequest"

     { "outboundMessageRequest":
       { "address": [ "+351915661333", "+351919876541" ],
         "clientCorrelator": "567895",
         "outboundMMSMessage": {
           "priority": "High",
           "subject": "Test MMS to you" },
    "receiptRequest":{
      "notifyURL":"http://www.yoururl.here/notifications"},
    "senderAddress": "+351915661336","senderName": "Me" } }

------------------------------a83abfca62f9
Content-Disposition:         form-data;         name="vcard";
filename="vcard_test.vcf"
Content-Type: application/octet-stream

BEGIN:VCARD
```

Snippet 11 - Example MMS send request body (part 1)

```
     VERSION:2.1
     N;LANGUAGE=pt:Doe;John
     FN:John Doe
     ADR;WORK;PREF:;;;Somewhere;;;Portugal
     LABEL;WORK;PREF:Somewhere
     X-MS-OL-DEFAULT-POSTAL-ADDRESS:2
     EMAIL;PREF;INTERNET:john.doe@mail.pt
     X-MS-OL-DESIGN;CHARSET=utf-8:<card
xmlns="http://schemas.microsoft.com/office/outlook/12/electronicbu
sinesscards"    ver="1.0"    layout="left"    bgcolor="ffffff"><img
xmlns=""  align="fit"  area="16"  use="cardpicture"/><fld  xmlns=""
prop="name"   align="left"   dir="ltr"   style="b"   color="000000"
size="10"/><fld   xmlns=""   prop="blank"   size="8"/><fld   xmlns=""
prop="email" align="left" dir="ltr" color="000000" size="8"/><fld
xmlns=""   prop="blank"   size="8"/><fld   xmlns=""   prop="addrwork"
align="left"   dir="ltr"   color="000000"   size="8"/><fld   xmlns=""
prop="blank" size="8"/><fld xmlns="" prop="blank" size="8"/><fld
xmlns=""    prop="blank"    size="8"/><fld    xmlns=""    prop="blank"
size="8"/><fld   xmlns=""   prop="blank"   size="8"/><fld   xmlns=""
prop="blank" size="8"/><fld xmlns="" prop="blank" size="8"/><fld
xmlns=""    prop="blank"    size="8"/><fld    xmlns=""    prop="blank"
size="8"/><fld   xmlns=""   prop="blank"   size="8"/><fld   xmlns=""
prop="blank" size="8"/></card>
     REV:20140325T110306Z
     END:VCARD

     ------------------------------a83abfca62f9--
```

Snippet 12 - Example MMS send request body (part 2)

By analyzing the request, one can see two distinct sections:

- A JSON body section, containing all the relevant message information, from the sender to the address to be sent to;
- An attachment containing a vCard (as per vCard 2.1 specifications[89]) or a JPEG image, in which case the "name" parameter should be set to "image" instead of "vcard";

In both sections, certain details must be emphasized, such as the Content-Disposition headers and the names. In contrast to the definition presented in the OneAPI specification, where the attachment would be sent in a sub-section of the multipart request, in this case one can see that the structure has been altered. This has been done in order to simplify implementation and the request body itself for easier analysis and data parsing. An example request can be seen depicted in Snippet 13.

```
curl -v -X POST -H 'Content-Type: multipart/form-data' -F
'outboundMessageRequest={"outboundMessageRequest": {"address":
["+351915661333"],"clientCorrelator":"567895","outboundMMSMessage"
:{"priority":"High","subject": "Test MMS to you"},
"receiptRequest":{"notifyURL":"http://www.yoururl.here/notificatio
ns/"}, "senderAddress": "+351915661336", "senderName": "Me"}}' -F
"vcard=@/home/user/Desktop/vcard_test.vf"
http://localhost:9090/WTE/resources/2013-12-
12/<applicationToken>/messaging/outbound/+351915661336/requests
```

Snippet 13 - Example MMS send request

In this example, a directory is used to locate the vCard to be sent. Again, if one were to send an MMS with a JPEG image, the attachment declaration would change from `"vcard=@/home/user/Desktop/vcard_test.vcf"` to a declaration pointing to an image file `"image=@/home/user/Desktop/image_test.jpg"`.

Response

A response to the request would be similar to the one in Snippet 14.

```
< HTTP/1.1 201 Created
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Mon, 28 Mar 2014 11:20:48 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"resourceReference": {"resourceURL":""}}
```

Snippet 14 - Example MMS send response

Again, the solution responds with the resourceURL parameter in blank, since no mechanism for message delivery state was implemented. In terms of message flow in the platform, this API request/response flow is exactly alike the previous text messaging flow in Figure 80, with the only deviation being in the content sent.

B.2.4) Notifications

Notifications are available via Webhooks, with clients registering a URL where they wish to receive call state notifications, as well as a long-polling mechanism. This feature-set refers to such mechanism and how this is mapped in the solution's API. As was the case in the authentication feature above, this long-polling mechanism was not presented in the original OneAPI specification. It was created to provide a fallback method for retrieving call notifications, in cases where the communication is possibly barred by network configurations, for example. Call state messages for the ongoing call sessions are stored in the server temporarily, with a lifetime of 60 seconds – after which they will simply be discarded. A third-party client is able to request the notifications of the call it triggered, identifying it by its call session identification string.

Request

Upon requesting the triggering of a call session, the third-party client may request its notifications by sending a GET request to the API resource in Snippet 15. At this time, and following long-polling execution flows, the request will hang until either (A) a message is available to be delivered to the client or (B) a timeout occurs, the latter happening if the request is not responded within 10 seconds. In order to continue to receive updates, the client must cyclically request notifications in this manner.

```
curl -v -X GET 'http://localhost:9090/WTE/resources/2013-12-12/<applicationToken>/notifications/callSessions/<callSessionId>/participants/<participantId>'
```

Snippet 15 - Example notification request

Response

The response to a notification request will be similar to the one in Snippet 16.

```
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Mon, 24 Mar 2014 16:49:30 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"CallStatus":"ringing","Caller":"+351913028148","To":"+351912552590","Called":"+351912552590","CallSid":"j80bam6ir1cpl7bq2cjc0q9tr","From":"+351913028148","Direction":"outbound-api","ApiVersion":"2013-12-12"}
```

Snippet 16 - Example notification response

With this response, the message was deleted from the server, as it waits for more notifications.

B.2.5) Make Call

The Make Call function relates to the triggering of a call between the server and the participants, connecting both via a media bridge allocated in the server.

Request

Make Call is available by making a POST request to the API, with the JSON body for a basic call session defined in Snippet 17 and a complete call session (with callback referencing) defined in Snippet 18.

```
{ "callSessionInformation":
  { "clientCorrelator": "104567", "participant":
    [ { "participantAddress":"+351910000001",
        "participantName":"Regardez Rust" },
      { "participantAddress":"+351910000002",
        "participantName":"Peter S. Errano" } ] } }
```

Snippet 17 - Example call triggering request body (basic call session)

```
{ "callSessionInformation":
  {"callbackReference":{"notifyURL":"http://www.yoururl.here/
  notifications" },
  "clientCorrelator": "304567",
  "originatorAnnouncement": "predefinedAnnouncement",
  "participant": [
     {"participantAddress":"+351910000001",
     "participantName":"Regardez Rust" },
     {"participantAddress":"+351910000001",
     "participantName":"Peter S. Errano" } ],
  "participantAnnouncement": "predefinedAnnouncement" } }
```

Snippet 18 - Example call triggering request body (complete call session)

This request can be done to the API, with the proper ContentType (application/json) via curl, an example depicted in Snippet 19.

```
curl -v -H 'Content-Type:application/json' -X POST
http://localhost:9090/WTE/resources/2013-12-
12/<applicationToken>/thirdpartycall/callSessions -d '{
'callSessionInformation': { 'callbackReference': {
"notifyURL":"http://localhost:8888/201302276d74ab1029fe03d4683028d
e0b36c54b-VaaSML_call1" }, 'clientCorrelator': '304567',
'originatorAnnouncement':'predefinedAnnouncement1ForOriginator',
'participant': [{ "participantAddress":"+351910000001",
"participantName":"Alexandre Brito" },{
"participantAddress":"+351910000002", "participantName":"Bruno
Cruz"}],'participantAnnouncement':'predefinedAnnouncement1ForParti
cipant'}}'
```

Snippet 19 - Example call triggering request

Response

Snippet 20 features the request's response, with a call placed between "+351910000001" and "+351910000002".

```
HTTP/1.1 201 Created
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Mon, 24 Feb 2014 10:50:04 GMT

{"resourceReference":{"resourceURL":"http://localhost:9090/WT
E/resources/2013-12-
12/<applicationToken>/thirdpartycall/callSessions/<callSessionID>"
}}
```

Snippet 20 - Example call triggering response

Call triggering flow

The interaction above results in the flow in Figure 81.

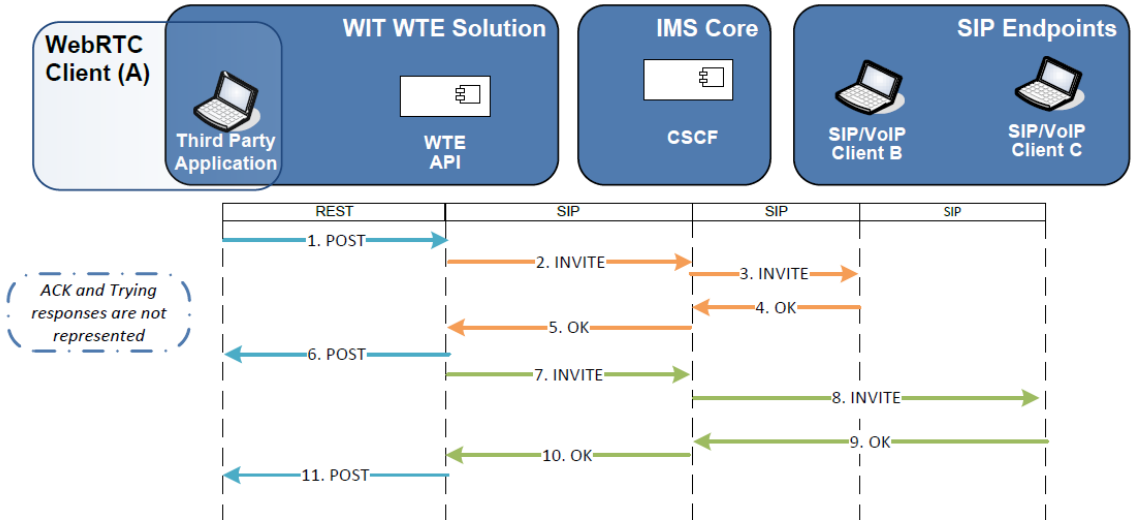

Figure 81 - Call Triggering flow

As one can see, a POST request is sent to the API, which in turn sends a SIP INVITE to the first participant. After receiving its response (with a 200 OK code), the AS notifies the third party application (if a notifyURL was used for subscription to these events) and then sends another SIP INVITE to the other participant. After receiving its response, the AS connects both users and media exchange is initiated.

B.2.6) Retrieve Call Information

The Retrieve Call Session Information feature allows a third party developer to get information regarding an on-going call session.

Request

This feature is available by placing a GET request to the solution, with the resource in Snippet 21 featuring both application token and call session identifier. This interaction does not reflect a SIP exchange, being confined to the retrieving of the required information from the internal server logic.

```
curl -v -X GET 'http://localhost:9090/WTE/resources/2013-12-12/<applicationToken>/thirdpartycall/callSessions/<callSessionId>'
```

Snippet 21 - Example call information retrieval request

Response

A request such has the above would result in a response similar to Snippet 22.

```
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 05 Mar 2014 10:48:29 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"callSessionInformation":{"participant":[{"participantAddres
s":"+351910000001"},{"participantAddress":"+351910000002"},{"parti
cipantAddress":"+351910000003"}]}}
```

Snippet 22 - Example call information retrieval response

B.2.7) End Call

The End Call feature is related to the ending of a call session, not an individual participant's session. This is a function that is translated into the hang-up of the call.

Request

This feature is available by placing a DELETE request to the solution, with the resource featuring both application token and call session identifier, as per Snippet 23.

```
curl -v -X DELETE 'http://localhost:9090/WTE/resources/2013-12-
12/<applicationToken>/thirdpartycall/callSessions/<callSessionId>'
```

Snippet 23 - Example call ending request

Response

A request such has the above would result in a response similar to Snippet 24, all in all similar to the retrieval of a call session's information.

```
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 05 Mar 2014 10:50:31 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"callSessionInformation":{"participant":[{"participantAddres
s":"+351910000001"},{"participantAddress":"+351910000001"}]}}
```

Snippet 24 - Example call ending response

Hang-up Flow

The hang-up process depicted in Figure 82 is very similar to the call triggering flow, with a few minor changes.

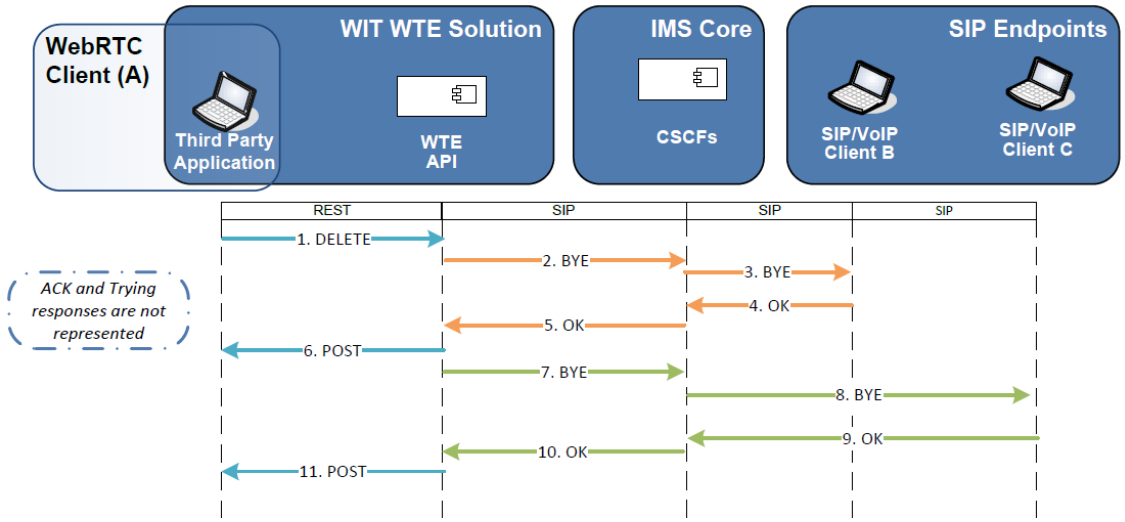

Figure 82 - Hang-up flow

As represented in Figure 82, a DELETE request is sent to the API, which in turn sends a SIP BYE to the first participant. Again, if a notifyURL was given, after receiving a 200 OK code, the AS notifies the third party application and sends a SIP BYE to the other participant. After receiving its response, the call is terminated with the allocated resources within the AS freed.

B.2.8) Add To Call

The Add To Call feature relates to the adding of a new participant to a call session, translating into the invitation of a participant and allocation to a specific conference.

Request

The feature is available by placing a POST request to the solution with a JSON body as the one defined in Snippet 25.

```
{"callParticipantInformation":
      {"clientCorrelator": "224567",
       "participantAddress":"+351912345678",
       "participantName": "Filipe Pinheiro"}
}
```

Snippet 25 - Example add to call request body

The curl in Snippet 26 is an example of the methods usage – note the presence of the content-type and both call session identifier/application token in the resource.

```
curl -v -X POST 'http://localhost:9090/WTE/resources/2013-12-
12/<applicationToken>/thirdpartycall/callSessions/<callSessionId>/
participants'                                                    -d
‘{"callParticipantInformation":{"clientCorrelator":
"224567","participantAddress":"+351912345678","participantName":"F
ilipe Pinheiro"}}’
```

Snippet 26 - Example add to call request

Response

The response will be similar to the example in Snippet 27.

```
     HTTP/1.1 201 Created
     < Server: Apache-Coyote/1.1
     < Content-Type: application/json
     < Transfer-Encoding: chunked
     < Date: Mon, 24 Feb 2014 10:51:04 GMT

     {"resourceReference":{"resourceURL":"http://localhost:9090/WT
E/resources/2013-12-
12/<applicationToken>/thirdpartycall/callSessions/<callSessionId>/
participants/<participantId>"}}
```

Snippet 27 - Example add to call response

During this interaction, a call should have been placed to "+351912345678", consequently connecting it to the on-going call session and exchanging the media.

Add To Call Flow

The hang-up process is very similar to the call triggering flow in Figure 81, with the only change being that only one call invitation results from the HTTP POST request.

B.2.9) Retrieve Participant Information

The Retrieve Participant Information feature allows a third party developer to get information regarding a participant in an on-going call session.

Request

This feature is available by placing a GET request to the solution, with the resource in Snippet 28 featuring application token and call session ID, in addition to participant ID. This interaction does not reflect a SIP exchange, being confined to the retrieving of the required information from the internal server logical (the listings that contain the calls and conferences).

```
curl -v -X GET 'http://localhost:9090/WTE/resources/2013-12-12/<applicationToken>/thirdpartycall/callSessions/<callSessionId>/participants/<participantId>'
```

Snippet 28 - Example participant information retrival request

Response

A request such has the above would result in a response similar to Snippet 29.

```
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 05 Mar 2014 10:49:18 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"callParticipantInformation":{"participantAddress":"+351913254657"}}
```

Snippet 29 - Example participant information retrieval response

B.2.10) Delete Participant from Call

The Delete Participant from Call Session feature enables the termination of a call-leg from a conference, returning participant information. This reflects the ending of a call for a participant, not the ending of the call for all parties involved.

Request

This feature is available by placing a DELETE request to the solution, with the resource in Snippet 30 featuring application token, call session ID and participant ID.

```
curl -v -X DELETE 'http://localhost:9090/WTE/resources/2013-12-
12/<applicationToken>/thirdpartycall/callSessions/<callSessionId>/participants/<participantId>'
```

Snippet 30 - Example participant deletion from call request

Response

A request such has the above would result in a response similar to Snippet 31.

```
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 05 Mar 2014 10:49:55 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"callParticipantInformation":{"participantAddress":"+351913243546"}}
```

Snippet 31 - Example participant deletion from call response

Participant Termination Flow

This individual call hang-up process is very similar to the call hang-up flow, with the only change being that only one SIP BYE is sent as a result to the HTTP DELETE request.

B.2.11) Hold/Resume Call

The Hold/Resume Call features relate to the triggering of a call hold/resume to a call session. With this method, one can put all participants of a given call session on hold and then resume the call on command.

This feature is not present in the OneAPI specification, having been implemented following the patterns shown in the previous methods. The parameter “callAction” is in fact new, chosen since no other parameter in the specification was close enough to properly describe the actions to be taken in the call.

Request

Both hold and resume features are available by placing a POST request to the solution, with the resource featuring both application token/call session identifier and a JSON body with the call action set to “hold” or “resume”, like depicted in Snippet 32.

```
{"callSessionInformation": {"callAction": "hold"} }
```

Snippet 32 - Example hold call request body

An example of its usage may be seen by the curl request in Snippet 33.

```
curl -v -H 'Content-Type:application/json' -X POST
'http://localhost:9090/WTE/resources/2013-12-
12/<applicationToken>/thirdpartycall/callSessions/<callSessionId>'
-d ‘{"callSessionInformation":{"callAction": "hold"}}’
```

Snippet 33 - Example hold call request

Response

The response will be similar to the example in Snippet 34.

```
HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Mon, 24 Feb 2014 10:51:04 GMT
```

Snippet 34 - Example hold call response

During this interaction, the call should have been put on hold or resumed, depending on its previous state (if the call was not on hold, or vice-versa, a 404 Not Found response will be triggered).

Hold/Resume Call Flow

The Hold/Resume process is very similar to a normal call triggering flow, with the participants being re-invited to participate in a call session set to a different conference room, where media will not be exchanged due to the changes made to the SDP exchanged in the re-invite.

C) User Provisioning API

This API can be used to remotely insert network users in the WTE solution, providing a way for operators that already use a back-office for creating new users in the HSS, to also communicate with the solution and store a synchronized version of the data, allowing users to make use of the same credentials across third party applications. The API has been modeled with the influence of OMA Push definition, allowing both XML/JSON content.

C.1) Features

As referred above, this API is only used to register network users in the solution, having the features below analyzed been implemented.

C.1.1) Request authentication

In its current state, the API proceeds to trigger authorization methods to all incoming requests, via an application token contained in the request body. The logic behind this decision was a design one: all applications that interact with the solution will be seen as third party – in this case, third party administrative applications. This means that an incoming request coming from this application (in this case, the operator's back-office), one that was previously published by an administrator, will go through an authorization mechanism in order to be processed.

C.1.2) User creation/update

Both user creation and update are available by making a PUT request to the API, with a JSON (Snippet 35) or XML (Snippet 36) body defined with all the mandatory fields – from application token to the user's desired password.

```
{ “subscription”: {
“applicationToken”: ”APP_TOKEN”
"user": {
     "impu": "ana.cruz@ua.pt",
     "msisdn": "1236547890",
     "firstName": "Ana",
     "lastName": "Cruz",
     "password": "testing" } } }
```

Snippet 35 - Example user creation request body (JSON)

```
<subscription>
<applicationToken>APP_TOKEN</applicationToken>
<user>
     <impu>ana.cruz@ua.pt</impu>
     <msisdn>1236547890</msisdn>
     <firstName>Ana</firstName>
     <lastName>Cruz</lastName>
     <password>testing</password>
</user>
</subscription>
```

Snippet 36 - Example user creation request body (XML)

This API can only be accessed by privileged applications (“administrative applications”, reserved for administration and not available from a user/developer marketplace). As such, it gives full control over users’ details, requiring only the email to stay unique and be maintained across updates.

Request

Example usage – properly setting content-type and method via a curl call -, can be seen in Snippet 37 and Snippet 38, with a JSON and XML body, respectively.

```
curl -v -H "Content-Type:application/json" -X PUT http://
localhost:9090/WTEPortal/userProvisionService -d
'{"subscription":{"applicationToken":"APP_TOKEN", "user":
{"impu":"ana.cruz@ua.pt", "msisdn":"1236547890",
"firstName":"Ana", "lastName":"Cruz", "password":"TestCruz"}}}'
```

Snippet 37 - Example user creation request (with JSON body)

```
curl -v -H "Content-Type:application/xml" -X PUT
http://localhost:9090/WTEPortal/userProvisionService -d
'<subscription><applicationToken>APP_TOKEN</applicationToken><user
><impu>ana.cruz@ua.pt</impu><msisdn>1236547890</msisdn><firstName>
Ana</firstName><lastName>Cruz</lastName><password>testing</passwor
d></user></subscription>'
```

Snippet 38 - Example user creation request (with XML body)

Response

An example response to a request with a JSON body can be seen in Snippet 39. Note that the response is a 200 OK HTTP code, signifying that the request was received, processed, and the user was created. If, for some reason, the body structure was invalid or the given application token was rejected because the application was not previously authorized to make such actions, the response would be a 400 Bad Request and a 401 Unauthorized, respectively.

```
> PUT /WTEPortal/userProvisionService HTTP/1.1
>      User-Agent:      curl/7.22.0      (x86_64-pc-linux-gnu)
libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host: 10.39.40.177:9090
> Accept: */*
> Content-Type:application/json
> Content-Length: 196
>
* upload completely sent off: 196out of 196 bytes
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Length: 0
< Date: Wed, 26 Feb 2014 10:01:00 GMT
```

Snippet 39 - Example user creation response

Processing flow

The diagram in Figure 83 demonstrates the resulting processing flow from the interaction above. As one can see, the API will check whether the request is well formed (if all the necessary fields are present and the package syntax is correct, among others), sending an HTTP 400 (Bad request) code otherwise. After this, the application token is checked, validating whether the application sending the request is authorized do to so. If not, the response will be an HTTP code 401 (Not authorized). Finally, only if the above conditions are met is the request processed, with the database registering a new user entry and informing the user of the creation (HTTP 200 OK response).

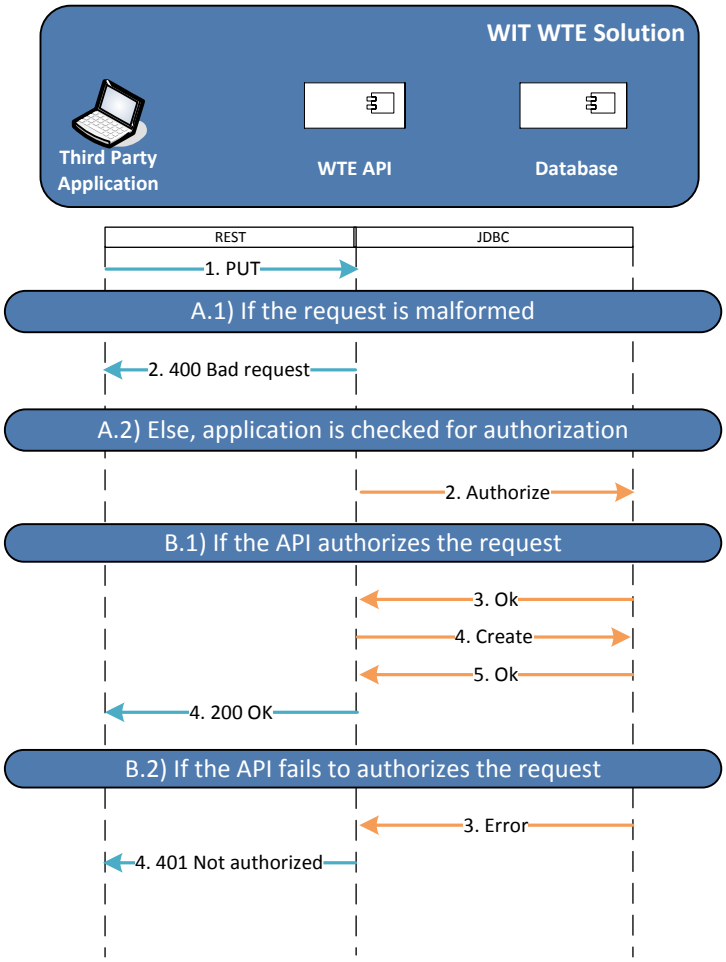

Figure 83 - User creation flow

C.1.3) User deletion

User deletion follows a similar approach, through a POST request with fewer mandatory fields. In Snippet 40 and Snippet 41, one can see these fields for a JSON and XML request.

```
{ "subscription": {
     "applicationToken": "APP_TOKEN"
     "user": {
          "impu": "ana.cruz@ua.pt",
          "msisdn": "1236547890",
          "password": "testing" } } }
```

Snippet 40 - Example user deletion request body (JSON)

```
<subscription>
<applicationToken>APP_TOKEN</applicationToken>
    <user>
        <impu>ana.cruz@ua.pt</impu>
        <msisdn>1236547890</msisdn>
        <password>testing</password>
    </user>
</subscription>
```

Snippet 41 - Example user deletion request body (XML)

Request

The curl examples are also very similar to the ones presented before and can be seen depicted in Snippet 42 and Snippet 43.

```
curl -v -H "Content-Type:application/json" -X POST http://
localhost:9090/WTEPortal/userProvisionService -d
'{"subscription":{"applicationToken":"APP_TOKEN", "user":
{"impu":"ana.cruz@ua.pt", "msisdn":"1236547890",
"password":"testing"}}}'
```

Snippet 42 - Example user deletion request (with JSON body)

```
curl -v -H "Content-Type:application/xml" -X POST http://
localhost:9090/WTEPortal/userProvisionService -d
'<subscription><applicationToken>APP_TOKEN</applicationToken><user
><impu>ana.cruz@ua.pt</impu><msisdn>1236547890</msisdn><password>t
esting</password></user></subscription>'
```

Snippet 43 - Example user deletion request (with XML body)

Response

Again, a curl such as the above would yield a response similar to Snippet 44.

```
> DELETE /WTEPortal/userProvisionService HTTP/1.1
>      User-Agent:      curl/7.22.0      (x86_64-pc-linux-gnu)
libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host: 10.39.40.177:9090
> Accept: */*
> Content-Type:application/json
> Content-Length: 155
>
* upload completely sent off: 155out of 155 bytes
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Length: 0
< Date: Wed, 26 Feb 2014 10:27:42 GMT
```

Snippet 44 - Example user deletion response

Processing flow

The processing flow is, all in all, similar to the flow in the user creation/update topic -Figure 83 -, with the difference being that from the database interaction results the erasing of the users' details.

C.1.4) HTTP responses

The HTTP codes on Table 20 are provided as responses to the requests made.

HTTP Code	Definition	Usage
200	Ok	To signal the execution of a request
400	Bad request	When a request was mal-formed
401	Unauthorized	When the request comes from an unauthorized source
404	Not found	When the processing of a request, issued to modify or delete a user, does not find the specified user
405	Method not allowed	When the request calls for an unsupported method
415	Unsupported media type	When the request features an unsupported content body

Table 20 - HTTP response codes

D) User provisioning solutions

In terms of user provisioning to use the platform's services, multiple approaches could be taken, each with its own set of advantages and disadvantages. However, the goal is to provide the user with the capability of using only one set of credentials – the one used to register in the IMS network.

D.1) Welcome notice

One of the approaches could be to send a welcome notice[90], upon first registration, as depicted in Figure 84.

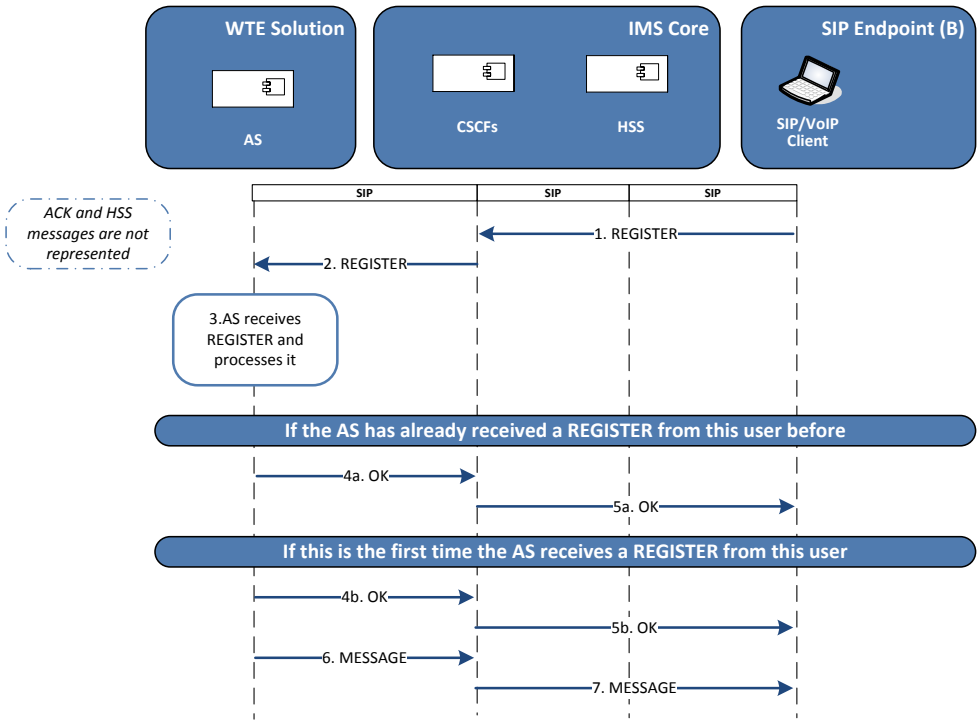

Figure 84 - Welcome notice

A welcome notice is no more than a message sent to the user upon first registration on the network. A common industry solution, this mechanism would, however, fail the goal proposed of using only one set of credentials, imposing a client-side dependency on text message support.

D.2) Third party registration

A second viable approach could be through third party registration via the application server, as seen in Figure 85.

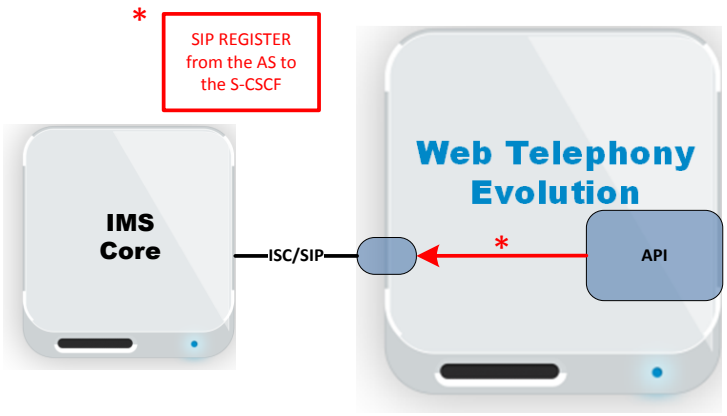

Figure 85 - Third party registration

This would reach the proposed goal, in addition to not demanding credentials to be stored in the platforms database, a security plus. Nonetheless, the AS would still be required to handle the added user session (one session for endpoint registration and another for API login) and the question remains on how the IMS would deal with the same IMPU being used. The OpenIMS core, for example, although capable of handling this situation, becomes unstable upon its occurrence.

D.3) Diameter/Cx integration

The only true integration considered, provisioning could be made via Cx integration, where the credentials sent for authentication would be checked against does in the networks HSS using the Diameter protocol.

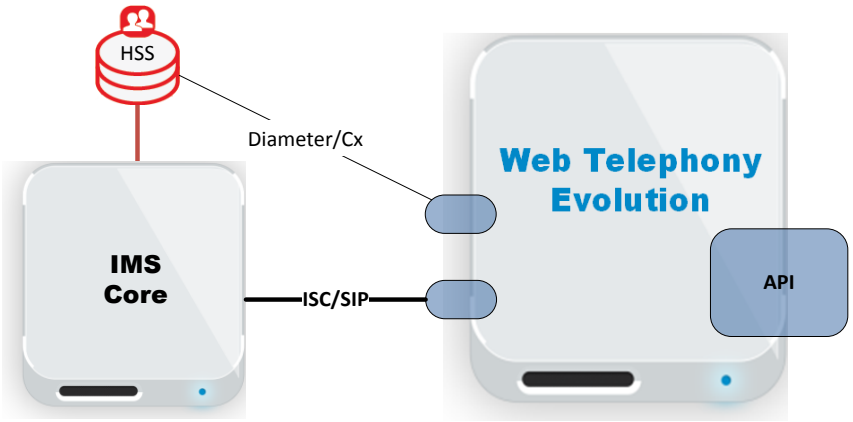

Figure 86 – Diameter/Cx integration

Again, among the approaches discussed in this annex, Figure 86 depicts the only true integration, with the credentials not being stored in the platforms database and with no extra consequences for the solution. This is not a perfect solution: excluding the added complexity of the implementation this mechanism would demand, security and network restrictions could eventually interdict access to this interface.

E) Notification mechanisms solutions

In this annex, alternative methods for notification handling will be analyzed, reviewing methods such as push or pull mechanics. Push or server push, refers to the transfer of data, with the communication started from the server to the client. Pull, by contrast, is when the client inquires information from the web server, with it responding with the desired information. There are several advantages and disadvantages to both:

- Push requires an open connection, where this pathway is used to send the information to the client. Consequently, this requires open ports from both server and client sides, creating problems in terms of network usage (especially in mobile devices) and port limitation;
- Pull requires constant multiple requests to the server which also creates problems in terms of network usage and bandwidth, but leaves out port limitations.

E.1) Push alternatives - WebSockets/HTTP Streaming

WebSockets or HTTP streaming is where a connection is established between client and server, allowing the latter to push notifications to the client. This results in real-time communication between the parties. The difference between both is in the initial handshake, which is repeated (in HTTP) for every message sent. In both cases, a bottleneck occurs in terms of concurrent connections and the opening of sockets would mean an increase in developer complexity for the API.

E.2) Pull alternatives - Long polling

Long polling is where a connection is cyclically opened between the client/server until a certain events occurs. Upon these events, data is sent and the connection is closed. This can be more efficient if the frequency of messages is low since, if you have two messages in the server awaiting client request, the second one will only be sent after the first. Comparing to the previous alternative, network management is less complex, since 1:1 port mapping of the server to the client would not be required. This also diminishes required resources - a 1:1 mapping would mean the reservation of a processing thread per user. Unfortunately, the problem with concurrent messages still stands but can be mitigated with efficient server-side logic and framework usage.

F) Testbed components and software

	Description
CPU	Intel Pentium Dual CPU T2310 @1.46 Ghz
Memory	3 GB
HDD	Speed: 50 MB/s
Network	Ethernet@1000 Mbps
OS	Ubuntu 12.04 LTS (32 bit)

	Description
CPU	Intel Pentium i7 CPU M60 @2.67Ghz
Memory	8 GB
HDD	Speed: 50 MB/s
Network	Ethernet@1000 Mbps
OS	Windows 7 Professional (64 bit)

Software	Version
OpenIMS Core	R1186
Sippy RTPproxy	v1.2.1
WIT Communications Application Server	v3.7
Wowza Media Server	v3
Sun JVM	v1.6.0_45
Apache Tomcat	v7.0.29
Mobicents	v2.0.0
MySQL	v5.5.35
PostgreSQL	v9.1.13
MTS (Multi-Protocol Test Suite)	v5.8.3

G) OpenIMS modification

The following modifications were done across the pcscf.cfg, icscf.cfg and scscf.cfg files of the IMS core, in order to maximize throughput:

```
7     # ----------- global configuration parameters ------------------------
8
9     debug=0
10    log_stderror=no
11    memlog=0
12    sip_warning=no
13
14    fork=yes
15    children=16
```

Snippet 45 - OpenIMS modification for evaluation

H) MTS default proprieties

The values defined in Table 21 were used in all MTS-involved test-cases.

File	Property	Default	Value
tester.properties	logs.STORAGE_LOCATION		FILE
tester.properties	stats.GENERATE_CHARTS_PICTURES		true
udp.properties	socket.RECEIVE_BUFFER	8192	1048576
udp.properties	socket.SEND_BUFFER	8192	1048576

Table 21 – MTS default configuration changes

This values define the storage location for the logs (defined here as “FILE”, meaning that logging will be stored in separate files in the client’s machine), chart generations (defined here as “true”, in order to generate delay charts, among others statistics) and general network buffer parameters (UDP buffers set to an higher value that the defaults, in order to maximize performance).

I) P-CSCF error and modification

The issue was located in the sdp_util.c file (present in the pcscf module of the IMS core), where one can find the following code:

```
975   /*
976    * Alter IP. Don't alter IP common for the session
977    * more than once.
978    */
979   if (c2p != NULL || !c1p_altered) {
980       body1.s = c2p ? c2p : c1p;
981       body1.len = bodylimit - body1.s;
982       if (alter_mediaip(msg, &body1, oldip, pf, newip, pf1, 0) == -1)
983             return -1;
984       if (!c2p)
985             *c1p_altered = 1;
986   }
```

Snippet 46 - OpenIMS's SDP utilities module

This will always return “false” since line 979 is missing an asterisk before `c1p_altered`. This line was changed to:

```
979    if (c2p != NULL || !*c1p_altered) {
```

Snippet 47 - SDP utilities modification

This modification corrected the issue and enabled the proper execution of RTPproxy enabled signalling flows.