Video-conference system based on open source software

Emanuel Frederico Barreiros Castro da Silva

Dissertation submitted to obtain the Master Degree in Information Systems and Computer Engineering

Jury
Chairman: Prof. Doutor Joao Paulo Marques da Silva
Supervisor: Prof. Doutor Nuno Filipe Valentim Roma
Co-supervisor: Prof. Doutor Pedro Filipe Zeferino Tomas
Member: Prof. Doutor Luis Manuel Antunes Veiga

June 2012
Acknowledgments

I would like to thank Dr. Nuno Roma for the patience in guiding me through all the research, implementation and writing phase of this thesis. I would also like to thank Dr. Pedro Tomás for the Latex style sheet for this document and for the final comments and corrections on this thesis.

I would also like to thank all my colleagues that helped me and supported me in hard times when i though it was impossible to implement the proposed work and was thinking in giving up. I would like to mention: João Lima, João Cabral, Pedro Magalhães and José Rua for all the support!

João Lima and João Cabral were also fundamental for me, since they discussed different possibilities in the proposed solution with me and gave me fundamental support and feedback on my work! You guys were really the best friends i could had in the time being! I do not have enough thank words for you!

André Mateus should also be mentioned due to all the patience and support for installing, uninstalling, installing, uninstalling and installing different Virtual Machines with Linux some more times.

I would like to thank my family, specially my mother, for all the support and care they gave me for almost two years.

Last but not least, i would like to thank my girlfriend, Rita Viegas, for all the support, kind words and incentive attitude she gave me in the final steps of finishing this thesis. All the strength that it was passed to me was a great value for me to keep working hard.
Abstract

This thesis proposes a free video conference solution for multi-room presentation sessions. It considers a system with a speaker, a local audience and a remote audience.

Even though there are many, well developed, video conference solutions that can perform this task they are typically expensive. The objective of this thesis is to develop a free solution, supported on low cost peripherals and computers, that can be extended to meet the requirements of the user.

The requirements set for this thesis are: dual view capabilities (to support a conference or presentation); bidirectional audio and video between the two rooms; a support on laptops hardware. This is achieved by using open source software for the interface, streaming and call management features. Moreover, it is also a goal of the proposed solution to communicate with current video conference solutions by using call signaling standards.

The implemented solution, can be run in a Linux operating system and has a web interface, where the multimedia streams are played and the application is controlled.

Keywords

Video conference, presentation session, call signaling protocols, multimedia streaming, acquisition and reproduction of media data.
Resumo

 Esta tese tem como objectivo propor uma solução de video conferência para sessões de apresentação em diferentes salas. Esta solução considera um sistema com um orador, uma audiência local e uma audiência remota.

 Mesmo existindo muitas soluções de vídeo conferência que executam esta tarefa de forma irrepreensível, estas soluções são tipicamente caras. O objectivo desta tese é desenvolver uma solução grátis, suportada em periféricos e computadores de baixo custo.

 O conjunto de requisitos para esta tese são: a capacidade de enviar e receber dois canais de vídeo (para o orador/audiência e para o conteúdo da apresentação), a capacidade de enviar e receber um canal de áudio (para o orador/audiência) e a capacidade de poder ser executada num computador portátil (utilizando os seus periféricos). Além disso, a capacidade de poder comunicar com outros sistemas de vídeo conferência é também um objectivo da solução proposta neste documento.

 A solução implementada pode ser executada no sistema operativo Linux e tem uma interface web onde serão apresentados os vídeos do orador/audiência (em conjunto com o audio) e do conteúdo da apresentação.

Palavras Chave

Video conferência, sessão de apresentação, protocólos de estabelecimento de chamada, streaming de multimédia, aquisição e reprodução de dados media.
Contents

1 Introduction ........................ 1
   1.1 Context ........................................... 2
   1.2 Application Domains ....................... 5
   1.3 Objectives ........................................ 6
   1.4 Contributions ............................ 7
   1.5 Structure of the thesis .................. 7

2 Supporting Technologies .................. 9
   2.1 Protocols/Standards Overview ........... 10
   2.2 Call Signaling Protocols ................. 11
      2.2.1 Session Initiation Protocol (SIP) ....... 13
      2.2.2 H.323 ...................................... 16
   2.3 Audio and Video Codecs ... 19
      2.3.1 Audio ..................................... 19
      2.3.2 Video ..................................... 20
   2.4 Streaming Protocols ....................... 21
      2.4.1 Real Time Protocol (RTP) ............... 23
      2.4.2 Real Time Streaming Protocol (RTSP) ... 24
      2.4.3 Hypertext Transfer Protocol (Hypertext Transfer Protocol (HTTP)) .... 27
   2.5 Summary ........................................ 28

3 Software Development Frameworks and Libraries 31
   3.1 Video Conference Frameworks ............ 32
      3.1.1 Portable Tools Library (PTLib) .......... 32
      3.1.2 H.323 Plus ................................ 34
      3.1.3 Open Phone Abstraction Layer (OPAL) ... 34
   3.2 Multimedia Streamer Frameworks .......... 36
      3.2.1 FFServer .................................. 37
      3.2.2 GStreamer .................................. 38
      3.2.3 Flumotion .................................. 40
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3 Summary</td>
<td>42</td>
</tr>
<tr>
<td>4 Current Solutions</td>
<td>43</td>
</tr>
<tr>
<td>4.1 Hardware Solutions</td>
<td>44</td>
</tr>
<tr>
<td>4.1.1 Aethra AVC 8400</td>
<td>44</td>
</tr>
<tr>
<td>4.1.2 Tandberg 990/880/770</td>
<td>46</td>
</tr>
<tr>
<td>4.1.3 Polycom QDX 6000</td>
<td>48</td>
</tr>
<tr>
<td>4.2 Software Solutions</td>
<td>49</td>
</tr>
<tr>
<td>4.2.1 VCON vPoint HD</td>
<td>50</td>
</tr>
<tr>
<td>4.2.2 Skype</td>
<td>51</td>
</tr>
<tr>
<td>4.2.3 Ekiga</td>
<td>51</td>
</tr>
<tr>
<td>4.2.4 Asterisk</td>
<td>52</td>
</tr>
<tr>
<td>4.2.5 Adobe Connect</td>
<td>53</td>
</tr>
<tr>
<td>4.2.6 Microsoft Lync</td>
<td>54</td>
</tr>
<tr>
<td>4.3 Summary: Hardware/Software Solutions Comparison</td>
<td>56</td>
</tr>
<tr>
<td>5 Proposed Architecture</td>
<td>59</td>
</tr>
<tr>
<td>5.1 System Architecture</td>
<td>60</td>
</tr>
<tr>
<td>5.2 Software Frameworks</td>
<td>62</td>
</tr>
<tr>
<td>5.2.1 Call Manager</td>
<td>63</td>
</tr>
<tr>
<td>5.2.2 User Interface</td>
<td>64</td>
</tr>
<tr>
<td>5.2.3 Multimedia Streamer</td>
<td>65</td>
</tr>
<tr>
<td>5.3 System description and framework integration</td>
<td>67</td>
</tr>
<tr>
<td>5.3.1 Opal and Flumotion Integration</td>
<td>67</td>
</tr>
<tr>
<td>5.3.2 OPAL and Interface Integration</td>
<td>70</td>
</tr>
<tr>
<td>5.3.3 Flumotion and Interface Integration</td>
<td>71</td>
</tr>
<tr>
<td>5.4 Summary</td>
<td>72</td>
</tr>
<tr>
<td>6 System Implementation</td>
<td>74</td>
</tr>
<tr>
<td>6.1 OPAL Call Manager Architecture</td>
<td>75</td>
</tr>
<tr>
<td>6.1.1 Call Manager Architecture</td>
<td>75</td>
</tr>
<tr>
<td>6.1.2 Call Flow</td>
<td>79</td>
</tr>
<tr>
<td>6.1.3 HTTP Server</td>
<td>80</td>
</tr>
<tr>
<td>6.1.4 Settings Storage</td>
<td>80</td>
</tr>
<tr>
<td>6.2 Flumotion Stream Server Configuration</td>
<td>81</td>
</tr>
<tr>
<td>6.2.1 Server Configuration</td>
<td>81</td>
</tr>
<tr>
<td>6.2.2 Worker Configuration</td>
<td>82</td>
</tr>
<tr>
<td>6.3 HTML Interface</td>
<td>83</td>
</tr>
</tbody>
</table>
Contents

6.4 Summary ................................................. 84

7 System Evaluation .................................... 86

7.1 Parameters Definition .............................. 87

7.1.1 Measures ............................................. 87

7.1.2 Parameters ........................................... 87

7.2 Tests Definition ....................................... 88

7.2.1 Self-Contained Tests ............................... 89

7.2.2 Integration with Other Video Conference Solutions .................................................. 89

7.3 Results .................................................. 90

7.4 Solution's states ...................................... 91

7.5 Summary ................................................. 92

8 Conclusion ............................................... 94

8.1 Advantages ............................................ 95

8.2 Features ................................................ 96

8.3 Limitations ............................................. 96

8.4 Future Work ........................................... 97
## List of Figures

1.1 Video conference room: speaker’s room. Proj. CD1 displays the supporting media and Proj. V2 displays the remote audience. ......................................................... 3

1.2 Video conference room: attendees room. Proj. CD1 displays the supporting media and Proj. V1 displays the remote speaker. ......................................................... 4

2.1 Protocols/Standards Overview. ........................................................................ 10

2.2 Example telephony network. ........................................................................... 12

2.3 SIP Call signaling process. ............................................................................. 13

2.4 SIP BYE message. ....................................................................................... 14

2.5 H.323 Call signaling process. ......................................................................... 16

2.6 H.323 Call control process. .......................................................................... 18

2.7 Architecture for video streaming. .................................................................... 22

2.8 HTTP Request - Response communication model. ........................................ 27

2.9 HTTP Request - Response with an intermediary agent. ............................... 28

3.1 PTLib abstract architecture. ........................................................................... 32

3.2 H.323 Plus abstract architecture. .................................................................... 34

3.3 OPAL abstract architecture. .......................................................................... 35

3.4 FFServer architecture. .................................................................................. 36

3.5 GStreamer pipeline architecture. ................................................................. 39

3.6 Flumotion Planet Architecture. ..................................................................... 41

3.7 Flumotion pipeline architecture. ................................................................. 42

4.1 Aethra AVC 8400 inputs and outputs ............................................................ 44

4.2 Tandberg 990 device. ................................................................................... 46

4.3 Polycom QDX 6000 device. .......................................................................... 48

4.4 VCON vPoint HD device. ............................................................................ 50

4.5 Screenshot of Ekiga software. ....................................................................... 52

4.6 Adobe Connect screenshot. .......................................................................... 54

4.7 Microsoft Lync screenshot. .......................................................................... 55
List of Figures

4.8 Current solutions comparison table. ........................................... 56
5.1 Global system architecture .................................................. 60
5.2 High level solution architecture ............................................. 62
5.3 Abstract to Implementation .................................................. 62
5.4 Comparison table between the currently available call signaling frameworks. . . . 64
5.5 Comparison table between multimedia containers ........................................... 65
5.6 Comparison table between multimedia streaming servers ........................................... 66
5.7 Detailed solution's modules interaction ........................................... 68
5.8 Launching Flumotion Server flow ............................................. 69
5.9 Integration between the call manager and the Interface Modules ................. 71
5.10 Communication flow between call manager module and interface module ...................... 72
5.11 Integration between Flumotion and the Interface Module ........................................... 73
6.1 Detailed OPAL architecture .................................................. 76
6.2 Call flow state diagram ...................................................... 79
6.3 Settings storage module ..................................................... 81
6.4 Flumotion processing pipeline .............................................. 82
6.5 Solution's screenshot with two video sources ........................................... 83
7.1 Evaluation results table ..................................................... 90
7.2 Speaker side of the implemented solution ........................................... 92
7.3 Audience side of the implemented solution ........................................... 93
Introduction

Contents

1.1 Context ................................................. 2
1.2 Application Domains .................................. 5
1.3 Objectives ............................................. 6
1.4 Contributions ......................................... 7
1.5 Structure of the thesis ................................. 7
1. Introduction

1.1 Context

Video conference gives individuals the opportunity to communicate with each other in real time, using video and audio. Through the advance of hardware technologies and the higher bandwidth capacity of networks, the long distance communication has become more popular and much more valuable for individuals and organizations.

Today's panorama of video conference is submerged by several capable applications which offer great value for long distance communications. These applications substitute the need for individuals to be in specific physical places to attend a meeting, thus reducing time and costs.

Video conference applications and solutions add another layer of touch between individuals when communicating with each other, which cannot be achieved by using just a telephone. In fact, if well implemented, video conference systems are able to provide all participants an experience that makes them feel they are attending a real meeting [11]. Therefore, communication is facilitated, ideas become clearer, less time is consumed and traveling costs are reduced [46].

Presentations

The main problem with most of presentation solutions is that they are developed for communication between individuals, where everyone speaks and collaborates the same way for the call. This is different from a presentation. When attending a presentation there are two main attention subjects:

- Speaker
- Supporting Media

The speaker is the person which is leading the presentation. Often, the speaker uses supporting media to back up the information he is stating. The supporting media can be of any type. Formerly score cards, slides or even black-boards were used. Nowadays the most used media types are electronic slides and video. Even though they are all different types of media, they all serve as the same purpose: to support the speaker in the presentation of the information to be given, which is why they are as important as the speaker itself.

Video Conference Rooms

Video conference rooms aim to solve these issues. In these most of these rooms it is installed a setup of hardware devices that enable a speaker to make a presentation simultaneously to local and remote attendees. Remote attendants also need a video conference room, even though being slightly different from the speaker's video conference room. Thus, for this solution to work, two physically different video conference rooms with specific video conference equipment are
1.1 Context

Figure 1.1: Video conference room: speaker's room. Proj. CD1 displays the supporting media and Proj. V2 displays the remote audience.

needed. From now on, these two video conference rooms will be denoted the speaker's room and the remote attendees room (or audience's room).

The speaker's room, illustrated in figure 1.1 is equipped with hardware to transmit his presentation and the media supporting him. In this site, the hardware needed to acquire the media is:

- Video camera, to acquire the speaker;
- Microphone, to acquire the speaker's voice;
- Video camera, to acquire the supporting media (instead of a video camera, a multimedia device, such as laptop or a video grabber, may also be used to send the supporting media);
- Video projector, to reproduce the remote audience;
- Speakers, to reproduce the remote audience's questions;
- Video projector, to reproduce the supporting media to the local audience.

In the remote audience side (see figure ??), the room also needs specific hardware equipment to acquire and reproduce the presentation:

- Video camera, to acquire the audience (composed by the attendees);
- Microphone, to acquire the audience questions;
- Video projector, to reproduce the speaker;
- Video projector, to reproduce the supporting media;
- Speakers, to reproduce the speaker and the supporting media (only if the media has an audio media component).
1. Introduction

Beyond the acquisition of hardware, each room also needs a specific and rather expensive video conference equipment which is responsible for acquiring the multimedia signal from each hardware peripheral, and transmit it to the other room. It is also responsible for receiving the media streams from other video conference rooms. This means the video conference device is the main component of each video conference room. It joins all the required components and transmits/receives the media information to be acquired/reproduced to the audience. The main requirements of such video conference equipment are:

- Simultaneously send two custom video sources that should be defined by the user (eg. webcam and screen grabber);
- Send one custom audio source (e.g. microphone);
- Synchronize the two video streams with the audio;
- Receive two video streams;
- Receive one audio streams.

Video Conference Solutions for presentations

As it will be mentioned in chapter 4, there are many applications that implement the transmission of video and audio between individuals over the Internet Protocol (IP) network. They cannot be classified as video conference solutions, but rather videophone solutions. Videophone solutions enable two individuals to communicate using video and audio. They are the extension (using a single video stream) of the telephone. Instead, video conference solutions allow presentations to take place in different sites. They also differ from videophone applications by the hardware requirements (which was described in the previous section).
The problem is that the video conference solutions available today are expensive. They require specific hardware (webcams, video projectors, microphones, etc) and the actual device to establish and manage the video conference calls. The main objective of this work, is to propose a free video conference solution (for presentation sessions) which can run in any laptop and use low cost peripherals.

## 1.2 Application Domains

Most current laptops already have built-in microphones and web-cameras, making them suitable tools for video conference presentations. Either by staying at the speaker’s side or the attendees side, laptops can substitute the typical hardware existing in video conference rooms, thus significantly reducing the costs. Some use case areas for video-conference presentations will be presented next, stating the valuable benefits it can offer real life scenarios.

### Education and Research

Education already benefits greatly from video-conference solutions. Presentations happen very often in these environments, either in conference quorums or theme classes. In research environments information is crucial and communication is the best way to deliver recent known knowledge. Current video conference solutions require researchers and/or teachers to go to a special room, where all equipments is installed, in order to lead a presentation for a remote audience.

A portable video-conference solution could help in several tasks in the academic world. Tasks such as: judging a thesis, present a subject, teach a class or participate in a conference meeting could be done by just using a laptop and the video conference software.

### Health Care

The dissemination of the latest state of the art trends about health care is dominated by several congresses over the world. Congresses are organized by a specific health subject, in which doctor and interns try to gain more knowledge and specialize themselves. When a congress happens, doctors from all around the world are gathered together to share and attend presentations.

However, there may be situations when a doctor cannot be present to lead a presentation which would be very valuable to the attendees of the congress. Through the proposed solution, that doctor could make his presentation at home, by simply using his laptop. The doctors attending the presentation could be in a wide auditorium, where the congress takes place, watching the remote presentation and communicate with the speakers by making questions. The speaker would also see his audience from his laptop enabling a visual communication by both parts.
1. Introduction

Business

Frequently, information technology organizations which develop and produce software products need to schedule training sessions for their users. Instead of having to physically move to all the organizations that bought the software, they could schedule video conference sessions where the software specialist would give a presentation/class about the software to the users.

The speaker could use his laptop to show the slides (using a media grabber) and the laptop’s webcam to capture the presentation. He could also see and listen to the audience’s questions and be able to answer them. On the other side, the audience, could be attending the presentation in a proper video conference room, or if that was not possible, a laptop could also be used. It is assumed that at least a projector (or a big screen) would be used to ensure that every attendee was clearly watching the presentation.

1.3 Objectives

As it was seen before, there are many application domains for a portable and low cost video conference presentation solution. The work to be implemented, as discussed in the previous section, aims to fill this gap in video conference solutions. The main focus is to build a video-conference system that is generic. That means that it can be used in ordinary everyday equipment (eg. laptop), using common low-cost peripherals and running on most Operative Systems. Additionally it shall be compatible with most video-conference solutions already available, whether they are free or proprietary. All this will be accomplished by using open-source software, tools and frameworks. To summarize, the main objectives are:

- To develop a versatile and low cost video conference application that includes:
  - Dual video output and input streams: for the video streams of the speaker and the supporting media.
  - Single audio output and input stream: for the speaker or audience.

- The strict use of open source software (frameworks and codecs);

- Communication and interoperability with other existing video conference devices and systems;

- Usage of standard video and audio codecs.

- The possibility to run on any kind of personal computer (laptop or desktop) with common low-cost peripherals (microphones, webcams, etc).

The main objective is to build a video conference application which enables any individual to give a presentation from his laptop. Additionally, it will also enable an audience to attend a remote presentation without the need for a fully equipped video conference room. For example, if
the remote audience only has access to one video projector, the application could send both the speaker and the supporting media video streams to that projector, using a split-screen configuration. It is expected from the application to adapt to the minimum hardware available (a video projector and a laptop) in a room and still enable a presentation to take place. This way, a regular room can be transformed in a video conference room in a fast and cost effective solution.

It is also highly desirable that the solution will cost as low as possible. To achieve this, only be used open-source software will be used to implement the solution. Consequently, the solution will be implemented to run on linux operating system and use any kind of video/audio peripherals.

Since there are video conference rooms already constructed, it is valuable that this solution communicates flawlessly and seamlessly with them, using standard network communication protocols and codecs.

1.4 Contributions

This thesis's main contribution to achieve the defined objectives is to prove that it is possible to implement a free video conference application, that can be used in any laptop and using low cost peripherals, based on open source software.

This video conference application can be used for multi-room presentation sessions where the remote room could be using the same implemented solution or other video conference solution.

The main contribution that a video conference solution with these properties offers individuals, is that they are no longer attached to expensive video conference rooms. It can be possible to setup a conference site, using just a laptop and low cost webcams, anywhere and instantaneously.

1.5 Structure of the thesis

This document will be divided in several chapters that describe the different aspects of the research and implementation phases. The first chapters: Supporting Technologies, Software Development Frameworks and Libraries and Current Solutions will focus on the research part of this thesis. The Supporting Technologies chapter will describe the different standards that are worth discussing for the objectives defined, such as call signaling standards, codecs and streaming protocols. The Software Development Frameworks and Libraries will focus on describing the frameworks and software tools available for developers to use, when developing multimedia/video conference applications. The Current Solutions, will describe some of the current video conference and videophone applications.

The implementation phase, described in the Proposed Architecture and the System Implementation chapters describes what are the architectural decisions that were made and how they were implemented.
1. Introduction

Finally, in the Evaluation Methodology chapter, it will be described the parameters to evaluate the implemented solution and the tests that were executed to validate the solution.
2 Supporting Technologies

Contents

2.1 Protocols/Standards Overview ........................................... 10
2.2 Call Signaling Protocols .................................................. 11
2.3 Audio and Video Codecs ................................................... 19
2.4 Streaming Protocols ....................................................... 21
2.5 Summary .......................................................................... 28
In this chapter presents, some of the most relevant supporting technologies for video conference which will constitute the basis of this thesis. These supporting technologies include call signaling protocols for communication, audio and video codecs and multimedia streaming protocols.

### 2.1 Protocols/Standards Overview

To introduce the studied protocols and standards it is better to define them by categories and explain why they appear in this document. This section serves as a roadmap to the Supporting Technologies chapter.

A video conference application needs to transmit multimedia streams to the user and transfer multimedia streams through the network. In figure 2.1 it illustrated the standards and the protocols that will be described later in this chapter. The figure is organized in a way similar to the Open Systems Interconnection (OSI) model. This is done because of the nature of the these protocols and standards. While most of them are networked based protocols, the others were defined (video and audio codecs) because of the bandwidth limits imposed by the IP network. The standards and the protocols were divided in the following layers:

- **Presentation/Distribution Layer**: This layer introduces the protocols and standards that are used to distribute media streams to the user. HTTP and RTSP can be used to distribute the media streams received from a remote client to the user.

- **Session Layer**: Since this layer is a middleware between the presentation and the transport layers, all the call signaling and application logic is implemented in this layer. The Session Layer can be further subdivided in two categories:
  
  - **Call Signaling Protocols**: Call signaling protocols, like H.323 and SIP, need to imple-
ment all the video conference logic in order to:

1. Transfer audio and video over the network with other remote client.

2. Transmit audio and video streams received from the remote client to the local user.

H.323 and SIP standards manage multimedia calls between two or more terminals, therefore, they need to manage the handshake and the flow of media streams.

- **Audio/Video Codecs**: Audio and video codec standards define multiple ways of encoding raw media streams. These media streams are required to have different properties depending on what they are going to be used for. In video conference applications, audio and video codecs are used for two reasons: encode media streams (to transfer them over the network) and encode media streams (to present the multimedia stream to the local user). That is why audio and video codecs were separated in two. This chapter it will be described different audio and video codec standards that have different properties to cope with bandwidth/quality balance.

- **Transport Layer**: The transport layer is used to transfer media streams and protocol specific information over the network. In the addition to the User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) which are usually defined in this layer, a third protocol, RTP, will be presented. RTP is used to transfer live multimedia streams over the network to maximize the viewing Quality of Service (QoS).

### 2.2 Call Signaling Protocols

Call signaling protocols allow two terminals to connect together so that users who are using those terminals can collaborate. Once the collaboration is over, call signaling terminates the connection\[17\].

In this section it will be discussed point to point call signaling protocols, using just two endpoints, without gatekeepers and/or gateways. This is done so that only the core connection procedures will be showed, leaving unnecessary details behind. Nevertheless, to better understand the elements that integrate a typical communication network an example topology and its elements will be described. The procedures for call establishment will be showed for SIP and H.323, which are the most common network communication protocols for both video conference and audio communication.

**Network elements**

In figure\[2.2\] illustrates a network example composed by commonly seen elements in real video and voice communication scenarios. In this network example, there are several particularities worth mentioning:
2. Supporting Technologies

- Different network protocols: there can be several network protocols that can’t communicate with each other.

- Different Terminals: there are also different terminals available. (e.g. Personal computers, laptops or mobile phones).

- Gateways: used to glue all different terminals and network protocols together. A gateway is responsible for translating from one network protocol to another network protocol. They can also be used to communicate with a specific terminal in an unusual way.

SIP and H.323 call signaling protocols interconnect terminals so that users can collaborate by sending voice and or video streams. One of the problems that these protocols had to solve was the interconnection between terminals in different networks. In this case, there are three different networks: Integrated Services Digital Network (ISDN), Public Switched Telephone Network (PSTN) and the IP network. Without these call signaling protocols, it would be very difficult to address all terminals in different networks, using different addresses for each. Gateways are the solution to this problem. Their task is to translate addresses among different networks, thus interconnecting terminals. For example, if terminal 4, located in the IP network, wants to call terminal 1, located in Public switched telephone network (PSTN) network, both gateway 1 and 4 could translate the IP address to a E.164 address and redirect the packets to the desired terminal. Gateways are also used in residential scenarios to transport packets from the IP Network to analog terminals such as phones. Generally speaking a gateway acts as a glue between networks.

Figure 2.2: Example telephony network.
2.2 Call Signaling Protocols

If a terminal cannot understand the protocol messages, it needs some sort of translator, which in this case is called a gateway.

2.2.1 SIP

The main objective of SIP is the communication between multimedia devices [16]. SIP makes the communication possible thanks to two protocols: RTP/RTP Control Protocol (RTCP) and Session Description Protocol (SDP). The RTP Protocol, extensively described in later sections is used to transport multimedia data in real time, whereas SDP protocol is used to negotiate the call capabilities between the participants. Both protocols will be described in further detail latter in this section.

SIP call signaling operation is based on the exchange of SIP messages via UDP channels. Any exchange of SIP messages is organized in a request and response pair. A request can have many responses associated with it. Responses are divided in two categories: provisional responses and final responses. In SIP, a request can have a provisional response to indicate that the party still does not know the outcome of a request. A response is final if it ends a transaction (set of a request plus all its associated response messages), which is usually a success or failure message. Figure 2.3 shows an “INVITE” transaction between two endpoints. The “INVITE” message is sent when the user wants to initiate a session with other endpoint. Once the called endpoint receives the INVITE message, it starts sending provisional “Ringing” responses, which indicates the user is online and being alerted. When the user finally accepts the call, a final “OK” response is sent.

The INVITE transaction is also used to inform the called endpoint about the capabilities the caller endpoint has, or wants to collaborate with. This is called the SDP Offer. The SDP offer, sent
2. Supporting Technologies

Figure 2.4: SIP BYE message.

along with the “INVITE” message, enumerates the set of capabilities (media types and respective codecs) the caller demands to collaborate with. In the “OK” final response, it will be included an SDP answer with the set of capabilities the called endpoint has. Since the caller needs to know the capabilities of its peer, another message is needed to acknowledge the “OK” message for this three-way handshake, the “ACK” message. The “ACK” message is the final message before both endpoints start transmitting the media, (voice and/or video) and collaborate with each other via an RTP channel. When an user wants to finish the session, the “BYE” message is sent, as shown in figure 2.4. An endpoint which receives this message terminates the associated session. Therefore it stops sending and listening for media packets from its peer.

SDP

As it was previously mentioned, SDP is used to negotiate the call capabilities among its endpoints. It is important to note that SDP does not define a true protocol, but rather, a language for representing the key parameters that characterize a multimedia session. An SDP message consists of a set text lines in the form:

\[ \text{< type >} = \text{< value >} \]  \tag{2.1}

Where \text{<type>} is a single character, and \text{<value>} is a structured text whose format depends on \text{<type>}. There are several \text{<type>} fields defined in SDP, some are required others are optional. These \text{<type>} fields are divided into three levels of information which are always present in an SDP message:

- Session-level description
- Time description
2.2 Call Signaling Protocols

- Media description

Session-level description contains information that describe characteristics of the whole multimedia session. The required information fields which describe the session are: “protocol version”, “originator and session identifier”; and “session name”. Session-level’s fields identify information as the originator username, session id, network and address type.

Time description contains information about time-related aspects of the session. Such aspects relate to the “time the session is alive”, which is the only required field, and the “repeat time”.

Media description includes information to characterize the different media present in the multimedia session. The characterization, through the available fields, indicates information such as the protocol to be used for media transport, transport address, media types to be used, media formats, bandwidth and connection information.

Media Protocols

Once the session description is defined, through the use of the SDP protocol, and every detail of the media to be transmitted has been assimilated by the endpoints, a protocol to take care of the media transport is needed. Instead of just having one protocol for media transportation, as it occurs on the signaling phase, the fact that there are different kinds of media types does not enable SIP to have just one media protocol that would fit all of them. In a multimedia session, as supported by SIP protocol, there are different types of multimedia sessions. Multimedia session can embrace real-time media (voice and video), quasi-real-time media (instant messaging) and even other types of media (file transfer). The protocol used to transport them must address different requirements imposed by the type of media. For instance, in real-time communication it is desired that lost packets should not be retransmitted, since this would cause delay on the receiving endpoint. On the other side, if the media to be transmitted were instant messages it would not be practicable to not retransmit lost packets since it would change the original message since even small changes in text can alter its meaning (also the time requirements are not too strict, so there is no need to accelerate the package transmission). Due to this fact, SIP can be used to set up media sessions supported on the following protocols:

- RTP: It is an Internet standard for the transport of strict real-time data such as voice or video.

- Message Session Relay Protocol (MSRP): Covers the transport of messages related to a session. MSRP is also being used for image sharing between mobile devices.

- TCP: SIP allows the use of generic TCP, so software developers can extend media types.

- T.38 fax transmission over UDP: The ITU T.38 recommendation describes the media transport for sending fax messages over IP networks (FoIP) in real-time.
2. Supporting Technologies

Figure 2.5: H.323 Call signaling process.

Since the main objective of video conference is real-time media transport, only RTP will be further explained. The other protocols serve just as a reference that they indeed exist and are subject of SIP's context and why SIP's architecture was designed to support multiple transport protocols.

2.2.2 H.323

Call establishment in H.323 protocol is divided in two stages: call signaling and call control. Call signaling is divided in five steps from call initiation to call termination, which will be later described. Once the call establishment is completed, the call control stage is responsible for orchestrating the coordination between the two endpoints. The whole process is illustrated in figure 2.5.

Call Signaling stages

The following steps are characterized by a flow of messages between the caller and the called terminal. The sent messages in the call signaling process are defined by the H.225.0-Q.931 recommendation. Each H.225.0-Q.931 message is built using the Information Elements of its corresponding message. Most of the work done in this recommendation was the definitions of the fields in the User-to-User Information Element for each message.

1. Call initiation: In the call initiation phase, the caller opens a TCP connection with the called endpoint in which H.225.0-Q.931 messages are sent. Once the TCP connection is established, the caller sends a “setup” message to the other endpoint. This setup message
is composed of the caller’s transport address (IP and TCP port) and its goal is to start the call establishment between the two endpoints. (See message 1 in figure 2.5).

2. **Call proceeding**: Upon receiving the setup message, the called endpoint sends a “call proceeding” message to the caller endpoint. This message intent is to inform the caller that the called endpoint is online. If the caller endpoint does not receive any answer in a stipulated time (defined in the standard by a timeout of four seconds), it assumes the called endpoint is not online, (see message 2 in figure 2.5).

3. **Call alerting**: After sending the “call proceeding” message, the called endpoint must alert his user that a new call is awaiting for an answer. This can be done in numerous ways: ringing tone, window pop-up, flashing window, etc. Once the user accepts the call, by answering it, a message “alerting” is sent to the caller endpoint to inform that the call has been accepted, (see message 3 in figure 2.5).

4. **Call connection**: In this stage, another TCP connection is open, this time to send H.245 call control messages which will sustain until one of the users decides to end the call. At this point, the call has been established and multimedia streams can now begin, (see set of messages 4 in figure 2.5).

5. **Call termination**: When the connection is over or the called user does not answer the call, both endpoint sends a “release complete” message to the other. Once this is done, the endpoints closes both H.225.0-Q.931 and H.245 TCP connections and the call is officially terminated, (see set of messages 5 in figure 2.5).

**Call Control**

Once the call establishment is completed, the call control process initiates. As mentioned in figure 2.6, there are four steps each endpoint needs to make in order to transmit media streams. These steps are:

1. The endpoints send the master/slave priority and provide their capabilities set in order to determine the extent of the collaboration.

2. After receiving the previous message, both endpoints can independently determine who is going to be the master and the capabilities for the call. The capabilities of the call are defined by an intersection of the capabilities of the two terminals.

3. In this step both endpoints send messages to each other in order to open logical channel connections. These can be voice, video or text.

4. Once a logical channel connection is open, the transmitter can start sending media to that logical channel. This is done by all logical channels open in the previous step.
As mentioned earlier, call control is made based on H.245 recommendation [18]. H.245 specifies the call control protocol for H.323, which consists of a set of messages and procedures. The protocol's messages are divided in three categories: Capability Management, Channel Management and Channel Operation. These messages help both endpoints to know what media and hardware capabilities the other endpoint has. Thus, the H.245 protocol provides the functionality to enable multimedia endpoints with different hardware and software capabilities to collaborate with each other.

- **Capability Management** If one endpoint starts transmitting using a given video standard (e.g. H.263), it might not be decoded if the receiving endpoint does not have the capability to receive that specific video standard. Therefore, the transmitter needs to know the receiver's capabilities before starting the transmission of any media. The capability messages describe the multimedia capabilities of the endpoint to its peer. These messages also help to defining which endpoint is the master and which endpoint is the slave. Upon determining the master/slave configuration, the master is responsible for resolving any conflicts on shared resources. Capability management messages also help each endpoint to transmit its capability set to its peer endpoint. The capability set of an endpoint is defined by which multimedia streams it can decode/encode.

- **Channel Management** In case the receiver has limited computational capabilities, it might need to know the characteristics of the media that it will receive, so that it can load the
appropriate decoder software, allocate memory and perform other housekeeping functions in order to get ready to receive the media. The transmitter offers this functionality through Channel Management messages. Channel Management manage logical messages between endpoints, helping them to allocate all the capabilities required to receive the media.

- **Channel Operation** Channel Operation messages provide specific functionalities to a specified channel. Using these types of messages, the remote peer can perform several specific operations to a given channel, such as: changing the maximum bit rate, update video frame, sync video, etc. [17]

### 2.3 Audio and Video Codecs

This section briefly describes the most common open source codecs available to use in media applications. Codecs enable media streams to be encoded before being transmitted over the network taking less bandwidth than the original media stream. Once the encoded stream reaches its destination, the codec can decode it generating a media stream close to the original stream. The codecs will be divided in audio and video codecs.

#### 2.3.1 Audio

Audio codecs perform encoding and decoding operations on audio streams. The next paragraphs presents some free audio codecs commonly used in media codification.

**Global System for Mobile Communications (GSM)**

GSM was originally designed to be part of the GSM Mobile phone network as its encoder; however, since then, it has evolved to be a standalone codec which can be used in other applications. It is still used to encode speech over the network. GSM is a lossy codec that applies a prediction algorithm, using the previous samples to construct the current sample. GSM standards define four compression techniques to analyze and compress speech samples: full-rate, enhanced full-rate, adaptive multi-rate and half-rate. All these codecs have been optimized properly to regenerate the original samples transmitted over a wireless network [12] [41].

**Speex**

Speex is an Open Source patent-free audio compression format specifically designed to encode speech. Speex’s algorithm is based on CELP linear predictor and was conceived to compress audio speech at bitrates from 2 to 44 kbps. Besides that, Speex codec also features noise suppression, acoustic echo canceler, voice activity detection, packet loss concealment, and many

1 data is lost during the encoding process
2. Supporting Technologies

other. Speex can also aggregate narrowband (8 kHz), wideband (16 kHz) and ultra-wideband (32 kHz) compression in the same bitstream [44].

G.711

G.711 is a high bitrate (64 kbps) ITU standard codec based on Pulse Code Modulation (PCM). It has two versions: PCMA (A-law) and PCMU (U-law). In both versions the signal is sampled with a logarithmic quantizer. The difference is that the A-law provides more dynamic range when compared with the U-law resulting in a better suppression of artifacts. Since it does not use any kind of compression G.711 provides the best voice quality. The downside is that it requires more bandwidth to transmit the audio stream over the network [32].

G.722

G-722 is a standard for high-quality voice communications. Its encoder uses a sub-band adaptive differential pulse-code modulation algorithm at 48, 56 or 64 kbps with input samples of 16 kHz. The G.722 standard has a dynamic algorithm which depends on the available bandwidth. When a high bandwidth is available, a lower compression algorithm is used. Conversely when a low bandwidth is available, higher compression rate are adopted. [33]

2.3.2 Video

Video codecs perform encoding and decoding operations on video streams in order to reduce the bandwidth needed to transmit it over the network. In the next paragraphs some free to use video codecs will be presented.

H.261

H.261 is an ITU video coding standard that was designed for data rates which are multiples of 64 kbps. It was originally designed to transport video streams over ISDN networks using the RTP protocol. The encoding algorithm is based on the exploitation of inter-picture prediction, transform coding and motion compensation technologies. The encoding process uses blocks of 8x8 pixels to compute the resulting compressed video. H.261 codec supports two image resolutions: QCIF (144x176 pixels) and CIF (288x352 pixels) [26][35].

H.263

H.263 is an ITU video compression standard originally designed as a low bitrate compressed format for video conference. This standard was developed as an evolution of H.261, MPEG-1 and MPEG-2 standards. In addition to QCIF and CIF image formats (supported by H.261), H.263 also supports SQCIF (half resolution of QCIF), 4CIF (4 times resolution of CIF) and 16CIF (16 times
resolution of CIF). After the first version of H.263, two more versions were developed: H.263v2 (H.263+) and H.263v3 (H.263++) [26][39].

**MPEG-4 Part 10 / H.264 AVC**

MPEG-4 Part 10 / H.264 AVC (Advanced Video Coding) is considered the state of the art video compression standard. It was designed to support efficient and robust coding. The original goal was to offer similar functionality to earlier video standards, such as H.263+, but with better compression and improved support for reliable transmission [26][40].

**Theora**

Theora video codec, which was also developed by xiph (the same developers of Speex audio codec), is a free and open source video compression format just like the other described standards. Theora-encoded data is transported inside the transport layer. Theora is a variable bitrate, lossy video compression scheme, exploits chroma subsampling, block-based motion compensation and an 8-by-8 DCT block [45].

### 2.4 Streaming Protocols

Streaming is defined as the set of communication protocols used to transmit media over the network between two terminals. The sender is usually called the server and the receiver is usually called the client. The most important particularity in multimedia streaming is that the media to be transmitted should arrive as quickly as possible at the client side, making the streaming experience as seamless as possible. This does not mean that the source media must be generated in real-time. In fact, a streaming process can transmit a previous stored media (such as music and a film) but can also transmit a real-time generated media (such as a sports live broadcast, audio content from a live concert or even an audio stream from a VOIP call). The difference between streaming and downloading a media stream is that with streaming the client's player should play the media as soon as the first signal packets arrive (after the local buffer has enough media data to start transmitting), which is the opposite from downloading media. When downloading a media stream, the client can only start playing the media when it was all downloaded and locally stored. This also means that in a case of a media stream that has no end (it is being generated in real-time), the only option for transmitting it is by streaming it to the client end.

Recently, the advances that were made in computing technology, namely in processing power and higher bandwidth networks made it possible to the common user to provide real time multimedia streaming services over the internet. Even though the Internet protocol does not assure a very high QoS standard to guarantee that every streaming service will play with minimum delay and/or media losses, it is still possible to offer a real-time streaming solution with satisfactory results. In
In this section, it will be presented a model for multimedia streaming over the internet. The model comprises all the tasks that should be implemented by the streaming server, as well as the client.

In Figure 2.7 it is illustrated a model for the streaming architecture. The model can be described in 6 steps: Video/Audio compression, Application-layer QoS control, Continuous media distribution services, Streaming servers, Media synchronization mechanisms and Protocols for streaming media. All these steps will be described below.

1. **Video/Audio compression**: As it was referred in a previous section, Raw video/audio signals must be compressed before transmission for such purpose. Compression techniques are generally used in codecs to guarantee a balance between communication efficiency and quality.

2. **Application-layer QoS control**: To cope with varying networks conditions and different presentation quality levels requested by the users, the streaming procedure must implement QoS control techniques. These include congestion control and error control. Congestion control mechanisms are used to prevent packet loss and reduce delay. Error control techniques are used to improve multimedia presentation quality in the presence of packet loss.

3. **Continuous media distribution services**: Continuous media distribution services, such as the IP and network, provide the channels necessary to transport the multimedia data from the server to the client. Continuous media distribution services include network filtering, application-level multicast, and content replication.

4. **Streaming servers**: Streaming servers play a key role in providing streaming services. To
offer quality streaming services, streaming servers are required to process multimedia data under timing constraints. Also, they must feature time efficient broadcast services in order to distribute multimedia streams.

5. **Media synchronization mechanisms**: Media synchronization mechanisms are the set of features that distinguishes traditional data applications from multimedia data applications. This mechanism assures that when dealing with multiple media contents, the correspondent set of streams will be presented the same way they were originally captured. This is very important when dealing with video conference solutions: the speaker’s lips should match his voice so that the remote audience can perceive his speech clearly.

6. **Protocols for streaming media**: To transport multimedia data over the network from one end to the other it is mandatory to have protocols designed and standardized for this specific purpose. Network protocols provide services such as network addressing, transport and session control. The network addressing (e.g. IP) and transport (e.g. UDP) are protocols that exist for every kind of data transmission. For streaming media, a special set of features are needed in order to enable the transmission to run as fast as possible, having in account eventual packet losses and errors. This service has the name of session control. There are several protocols that handle session control for real-time media streaming such as the RTSP.

2.4.1 **RTP**

The design principles of IP assume that the network infrastructure is unreliable at any single network element or transmission medium and that it is dynamic in terms of availability of links and nodes. Moreover, IP is a connectionless protocol, which means that there is no handshake to establish end-to-end connection before transmitting data [13]. Due to this fact, IP networks produce undesirable effects when carrying media traffic:

- **End-to-End Delay**: The delay is caused by the processing delay at each endpoint plus the delay caused by the IP network itself. End-to-End delay has poor results when collaborating with others. Frequently users guess that their peer is not listening and they repeat the message. This behavior produces an avalanche effect in overlapped messages in both directions.

- **Packet Loss**: Packet Loss is caused by the unreliable nature of IP Network. The common approach to solve this problem is to retransmit the lost packet, but since the media to be transmitted has real-time constrains, the extra delay is unacceptable. Due to this fact, generally, lost packets are not retransmitted.

- **Out-of-sequence Delivery**: Another undesired effect of the IP network is that packets from
2. Supporting Technologies

one endpoint to another may take different paths. Since some path may take longer than others, the packets to be transmitted can reach its destination out-of-order.

- **Jitter**: The jitter effect is a variable delay imposed by the routers. The time it takes a router to process a packet depends on its congestion level, and this may vary during the session. This variation is called jitter. Jitter, as the end-to-end delay effect, produce difficulties in real-time communication. The difference is that jitter introduces a variable delay, while end-to-end delay has a fixed effect.

Naturally, the mentioned disturbances affect the stable delivery of media packets over the network. The main goal of RTP is to solve these undesirable effects of the IP network, providing end-to-end delivery services for data with real-time characteristics.

To solve IP network problems, such as packet loss and out-of-sequence delivery in a timely fashion, RTP runs on top of the UDP layer. Real time communications impose real time constrains that TCP cannot solve. Therefore, running on top of UDP enables RTP to implement packet loss and out-of-sequence prevention measures. This way, it can also have more control solving issues like jitter and end-to-end delay [27].

2.4.2 RTSP

The RTSP is an application level protocol to provide control over the transmission of data with real time constrains. RTSP provides an extensible framework to enable controlled and on-demand delivery of real-time multimedia data, such as audio and video. Sources of data can include both live data feeds and stored clips. This protocol is intended to control multiple data delivery sessions, providing the means for choosing delivery channels (such as UDP, multicast UDP and TCP) and delivery mechanisms based upon RTP.

The RTSP establishes and controls either a single or several time-synchronized streams of continuous multimedia. It does not actually deliver the streams itself, leaving this task to other transport protocols. The set of streams to be controlled is defined by a presentation description. There is no notion of an RTSP connection. Instead, a server maintains a session labeled by an identifier. An RTSP session is in no way tied to a transport-level connection, such as a TCP connection. In fact, during an RTSP session, an RTSP client may open and close many reliable transport connections to the server, in order to issue RTSP requests. Alternatively, it may use a connectionless transport protocol, such as UDP [10].

The protocol supports the following operations:

- **Retrieval of contents from the media server**: The client can request a presentation description via HTTP or some other method. If the presentation is being multicasted, the presentation description contains the multicast addresses and the ports to be used for the
2.4 Streaming Protocols

continuous media. If the presentation is to be sent only to the client via unicast, it is the client who provides the destination, for security reasons.

- **Invitation of a media server to a conference**: A media server can be “invited” to join an existing conference, either to play back media into the presentation or to record all or a subset of the media in a presentation. This mode is useful for distributed teaching applications. The several parties in the conference may take turns, by “pushing the remote control buttons.”

- **Addition of media to an existing presentation**: Particularly for live presentations, it is useful if the server can inform the client about additional media becoming available.

Each presentation and media stream is represented by an RTSP URL, e.g.,

```rtsp://media.example.com:8888/ex```

While media.example.com:8888 represents the Internet location of the server, the /ex part of the URL is the specific presentation the client wants to connect to. Since an RTSP server can manage multiple presentations at the same time.

The overall presentation and the properties of the media streams are defined in a presentation description file. The format of the presentation description file is not defined in the RTSP protocol. The client should obtain the presentation description file from other outside sources. The presentation description file contains a description of the media streams making up the presentation, including their encodings, language, and other parameters that enable the client to choose the most appropriate combination of media. In this presentation description, each media stream that is individually controllable by RTSP is identified by an RTSP URL, which points to the media server handling that particular media stream and names the stream stored on that server. Several media streams can be located on different servers; for example, audio and video streams can be split across servers for load balancing. The description also enumerates which transport methods the server is capable of.

RTSP handshake may be sent via a separate protocol, independent of the control channel. For example, RTSP control may occur on a TCP connection while the data flows via UDP. Therefore the data delivery continues even if no RTSP requests are received by the media server. Also, during its lifetime, a single media stream may be controlled by RTSP requests issued sequentially on different TCP connections. Therefore, the server needs to maintain the "session state" in order to be able to correlate RTSP requests with a stream. Many methods in RTSP do not contribute to the state. However, the following play a central role in defining the allocation and usage of stream resources on the server:

- **SETUP**: Causes the server to allocate resources for a stream and starts an RTSP session.

- **PLAY or RECORD**: Starts data transmission on a stream allocated via SETUP.

- **PAUSE**: Temporarily halts a stream without freeing server resources.
2. Supporting Technologies

- **TEARDOWN**: Releases the resources associated with the stream. The RTSP session ceases to exist on the server.

**Message Format**

RTSP is a text-based protocol which makes it easier to extend with optional parameters. The RTSP protocol is based on a request/response communication model between the client and the server. In this section it will be described in a briefly manner what is the main structure of those messages.

**Request Message** The request message is composed of a method, whose options are to be applied to the server which is represented by an RTSP URL. The request message format is the following:

```
request = Request-Line *(Request-Header) [message-body]
```

The most important section of the request message is the Request-Line parameter. This parameter is composed of the method to be applied, the RTSP URL of the server and the RTSP protocol version. The Request-Header contains other type of additional information such as the transport, session, and others. The message-body is an optional parameter and is usually used to send additional information to the server. An example of a request message is:

```
SETUP rtsp://audio.example.com/twister/audio.en RTSP/1.0
CSeq: 1
Transport: RTP/AVP/UDP;unicast;client_port=3056-3057
```

The above example can be translated to the following. The SETUP part is the Request Line or the method, the url is the server to which the method is to be applied and the protocol is the request header. The rest of the message is the message body.

**Response Message** After receiving and interpreting a request message, the server responds with an RTSP response message. The message has the following format:

```
request = Status-Line *(Request-Header) [message-body]
```

Again, the most important part of the response message is the Status-Line which represents the result of the method sent in the request message. The Status-Line consists of the used RTSP protocol, a numeric status code and the textual representation of the status code. The other parameters serve the same purposes as in the request message. An example of a response message is:

```
RTSP/1.0 200 OK
CSeq: 1
Session: 12345678
Transport: RTP/AVP/UDP;unicast;client_port=3056-3057;server_port=5000-5001
```

The first line of the example is the actual answer or the Status line. The other lines are the message body. Notice that there is no request header in this message, simply because it was not needed.
2.4 Streaming Protocols

2.4.3 Hypertext Transfer Protocol (HTTP)

The HTTP is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic and stateless protocol, which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers [25].

Being a request/response protocol, each HTTP client sends a request to the server in the form of a request method and an URI, followed by a MIME-like message. The MIME-like message contains request modifiers, as well as client information. The server responds with a status line including a success or error code, followed by a MIME-like message containing server informations and entity meta-information.

Most HTTP communications are initiated by a user agent and consist of a request to be applied to a resource on a server, as illustrated in figure 2.8. In the simplest scenario, this may be accomplished via a single connection between the user agent and the server. More complicated situations can occur when one or more intermediaries are present in the request/response chain, represented in figure 2.9.

Usually, there might be three kinds of intermediaries: Proxy, Gateway and Tunnel. A proxy is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI. A gateway is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server’s protocol. A tunnel acts as a relay point between two connections without changing the messages. Tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

HTTP communication usually takes place over TCP/IP connections. The default port is TCP 80 but other ports can be equally used. This does not mean that HTTP cannot be implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable

---

Figure 2.8: HTTP Request - Response communication model.

2 Multipurpose Internet Mail Extensions (MIME): Defines the standard format of textual mail messages on the Internet [21].
2. Supporting Technologies

![HTTP Request - Response with an intermediary agent.](image)

Figure 2.9: HTTP Request - Response with an intermediary agent.

Transport. Any protocol that provides such guarantees can be used [25].

When the connection is established using the underlining reliable transport protocol. HTTP messages follow a request/response model using MIME-like messages. The request message, which is always started by the user agent has the following form:

```
GET [resource] [HTTP Version]
```

A simple example of a HTTP request message is: "GET /example.htm HTTP /1.1". After the request is received, the server processes the request and answers with a response message. The response message has the following structure:

```
[HTTP Version] [Status Code] [Status Code Description]
```

There are several status codes which indicate the success or failure of the user agent’s request. The most common status codes are: "200 OK", "404 Not Found", "301 Moved Permanently", "500 Server Error". An example of a response from the server’s behalf could be: "HTTP/1.1 200 OK".

2.5 Summary

This chapter focused on describing the standards and protocols that are related to call signaling standards, multimedia codecs and streaming protocols.

The call signaling standards that were described: H.323 and SIP manage calls over the Internet. From the call initiation, capabilities negotiation, media transmission and call termination. The difference between these standards is that SIP is an older standard, thus less refined than H.323. The SIP standard is simpler and has less control over the media streams that are being transmitted, otherwise it performs as good as H.323.

It was also described several audio and video codecs. They are fundamental to a multimedia application, since they offer different codification configurations of video/audio quality and bandwidth requirements to transmit the media streams over the internet.
Finally, it was described some media streaming protocols such as RTP, RTSP and HTTP. These protocols offer the capability of streaming multimedia data over the internet or for local use (e.g. to present the multimedia data in the local environment).
2. Supporting Technologies
3

Software Development Frameworks and Libraries

Contents

3.1 Video Conference Frameworks ........................................... 32
3.2 Multimedia Streamer Frameworks ...................................... 36
3.3 Summary ........................................................................... 42
3.1 Video Conference Frameworks

In the following sections it will be described the current frameworks available for developers to build video conference applications. These frameworks aim to deliver the developer tools for implementing platform independent applications as well as implementing the most common call signaling standards described earlier.

3.1.1 PTLib

The PTLib [22] is a class library that has its roots in The Portable Windows Library (PWLib), a library to develop applications to be run on both Microsoft Windows and Unix operating systems. It has also been ported to many other operating systems. Since then, the library has grown to include many other classes that assist in writing complete multi-platform applications. So it became a Portable Tools Library and was renamed to PTLib. The development of the OpenH323 [5] and OPAL [29] projects, as a primary use of the PTLib, has emphasized the focus on networking, I/O portability, multi-threading and protocol portability. Mostly, the library is used to create high performance and highly portable network-centric applications.

The classes within PTLib are separated into two types: Base Classes and Console Components.

Base Classes

The Base Classes contain the essential support for constructs, such as containers, threads and sockets, that are dependent on platform specific features. All PTLib programs will require the Base Classes.

The base object “PObject” is the root object in PTLib, where all other classes in the library are a specialization of this class. This is represented in figure 3.1 PTLib also defines container objects, which are also included in object-oriented languages and help the developer to deal with sets of elements. In PTLib, the container classes include: PContainer, PArray, PList, PDictionary, PSet and PString. These classes implement very well known data structures in an operating system independent way.
3.1 Video Conference Frameworks

Input/Output classes are also present in PTLib so that the developer does not have to use native operating system functions. Instead, the developer should use the set of objects that PTLib provides. PChannel is the root class, from where every other I/O class derive. There are several channels available but the most important channels are: PSerialChannel, PFile, PStructuredFile, PTextFile, PMemoryFile, PFilePath and PSoundChannel.

Socket implementation is also in the PTLib’s scope. It defines an abstract class PSocket which serves as the base class for all other socket types: PUDPSocket (IP socket for UDP), PTCPSocket (IP socket for TCP), PICMSocket (IP socket using ICM) and PEthSocket (socket interface for raw Ethernet interfaces).

Another type of base classes is the implementation of processes and threads. PTLib offers classes for creating processes and threads as well as concurrent mechanisms like semaphores. The classes that are used to manage processes and threads are: PProcess, PThread, PThreadPool, PSemaphore, PMutex, PCriticalSection, PSyncPoint and PSafeObject. PTLib also offers several classes that normalize other operating system functions, e.g., PConfig (that provides persistent storage for program settings using a platform-appropriate mechanism), PTime, PTimeInterval or PTimer.

Console Components

The Console Components implement a functionality that is usually platform independent, and may not be required for all programs. In some platforms (notably Windows) the Base Classes and Console Components may be divided into discrete library archives. Other platforms (notably Unix platforms) combine all the code into a single library and rely on the linker to omit code that is not required.

The console component classes are divided in three categories: Protocol Classes, HTTP Classes and Miscellaneous Classes. Protocol Classes Implement various Internet-related protocols. Some of these are implemented within PTLib, while others require external libraries to support the offered classes: PPOP3, PFTP, PTelnetSocket, PSOAPClient.

HTTP classes implement the HTTP Protocol. The classes available for the developer include: PURL, PHTML and PHTTPFORM. These classes help sending and formatting HTML streams.

Finally the Miscellaneous Classes are a set of classes that will help the developer in several tasks or problems it encounters: PArgList (parse a command line passed to a console program), PRandom, PRegularExpression, PCypher (implementation of various code cyphers such as PMessageDigest5, PTEACypher, and PMessageDigestSHA1), PBase64, PWAVFile (implements a AIFF format WAV file), PDTMFDekoder (decodes Dual-Tone Multi-Frequency (DTMF) digits from a stream of PCM data) and PXML are some of the helper classes which PTLib implements.
3. Software Development Frameworks and Libraries

![H.323 Plus abstract architecture](image)

**Figure 3.2: H.323 Plus abstract architecture.**

### 3.1.2 H.323 Plus

H.323 Plus [24] is a protocol framework with aim of fully implementing the H.323 call signaling standard, with all the features that were described in a previous section. H323 Plus has a set of base classes in its API, illustrated in figure 3.2 that implement the whole mechanism behind the framework. The main available classes are:

- **H323Endpoint**: This class manages the H323 endpoint. An endpoint may have zero or more listeners to create incoming connections or zero or more outgoing connections. Once a connection exists it is managed by this class instance. The main aspect this class embodies is the set of capabilities supported by the application, such as the codecs and protocols it is capable of.

- **H323Channel**: This class describes a logical channel between the two endpoints.

- **H323Capability**: This class describes the interface to a capability of the endpoint, usually a codec, used to transfer data via the logical channels opened and managed by the H323 control channel.

- **H323Listener**: This class describes a "listener" on a transport protocol. A "listener" is an object that listens to incoming connections on the particular transport.

- **H323Transport**: This class describes a I/O transport protocol, such as TCP or UDP.

Through the creation of descendant classes and the override of functions, the developer can create logic and use the classes made available in this framework to develop video conference applications.

### 3.1.3 OPAL

The OPAL [29] is a telephony framework which implements the H.323 and the SIP standards. It incorporates audio codecs including G.711, GSM06.10, Speex and iLBC. The API defines several classes which are essential to build an application using this framework. Some of the most important classes, illustrated in figure 3.3 are:
3.1 Video Conference Frameworks

![Diagram of OPAL abstract architecture]

- **OpalManager**: This class is the central manager for OPAL. The OpalManager embodies the root of the tree of objects that constitute an OPAL system. It contains all of the endpoints that make up the system. Other entities, such as media streams, etc., are in turn contained in these objects. It is expected that each application would only ever have one instance of this class, and also descend from it to override call back functions.

- **OpalEndPoint**: This class describes an endpoint base class. Each protocol implementation creates a descendant of this class to manage its particular subsystem. Typically, this would involve listening to incoming connections and being able to set up outgoing connections. An endpoint will also have a default set of media data formats that it can support. Connections created by it would initially have the same set, but according to the semantics of the underlying protocol, they may end up using a different set. H323Endpoint and SIP Endpoint are descendants of this class.

- **OpalCall**: This class manages a call. A call consists of one or more OpalConnection, see below, instances. While these connections may be created elsewhere, this class is responsible for their disposal.

- **OpalConnection**: This is the base class for connections to an endpoint. A particular protocol will have a descendant class from this to implement the specific semantics of that protocols connection. The connection is also in charge of creating media streams.

- **OpalMediaFormat**: This class describes a media format, as used in the OPAL system. A media format is the type of any media data that is transferred between OPAL entities. For example, an audio codec, such as G.723.1, is a media format, a video codec such, as H.261, is also a media format.

- **OpalMediaStream**: This class describes a media stream as used in the OPAL system. A media stream is the channel through which media data is transferred between OPAL entities. For example, data being sent to an RTP session over a network would be transferred through
3. Software Development Frameworks and Libraries

a media stream.

- **OpalTranscoder**: This class embodies the implementation of a specific codec instance used to convert data from one format to another. An application may create a descendent of this class and override functions as required to implement a codec.

- **OpalListener**: This class describes a “listener” on a transport protocol. A “listener” is an object that listens for incoming connections on the particular transport. It is executed as a separate thread.

- **OpalTransport**: This class describes an I/O transport to a protocol. A “transport” is an object that allows the transfer and processing of data from one entity to another.

The classes described above are the main classes that make OPAL mechanisms work. OPAL offers implementations for several classes which are based on the ones described. This enables the developer to choose the appropriate classes to implement the aimed application logic.

3.2 Multimedia Streamer Frameworks

This section introduces the multimedia streaming frameworks, which offer multimedia streaming capabilities so that it can be used as part of a higher level software.
3.2 Multimedia Streamer Frameworks

3.2.1 FFServer

FFServer is a module of FFmpeg\(^1\) which acts as a streaming server for both audio and video streams. It supports:

- **Several live feeds**: a server instance can manage multiple feeds at the same time.

- **Multiple streams from the same feed**: a feed can be streamed with different configurations at the same time.

- **Streaming from files**: the FFServer can stream multimedia streams that are stored in a local file.

- **Time shifting on live feeds**: depending on the adopted buffer sizes, the user can seek to positions in the past on each live feed.

FFServer receives pre-recorded files or FFM streams from an FFmpeg instance as input. Then, it streams them over RTP/RTSP/HTTP. The FFServer configuration is specified in a configuration file. The configuration file specifies what are the input streams to be received from FFmpeg and the output streams to be streamed by the FFServer. It also specifies the global configurations of the instance server. Such configurations include: used port to receive and stream the multimedia content; maximum number of connections allowed; maximum bandwidth; etc.

Figure 3.4 illustrates a possible architecture of an FFServer instance. As it was previously referred, FFServer receives an FFmpeg stream that is transcoded in as many streams as it is defined in its configuration file. The transcoding phase is also done using FFmpeg instances, inside FFServer. The resulting streams are then broadcasted in a feed that gathers all of them. The streams are then played in specific media players.

Below, it is presented a sample configuration file from an FFServer instance. This configuration file has four important parts which are worth mentioning:

1. The first part is the global configuration. In this part it is stated the port number that is going to be used by the server to stream and receive the multimedia content. It is also defined the maximum number of connections, clients and bandwidth.

2. The feed part is the definition of the input streams. The feed tag defines the file to be used as a buffer and the maximum size the buffer can grow. As a security measure, it can also be defined from what IP addresses the FFServer accepts connections to receive streams from FFmpeg (or other medium).

\(^1\)FFmpeg is a media format converter that can also grab from live multimedia sources.
3. Software Development Frameworks and Libraries

3. The multimedia content to be streamed is defined in a stream tag. The stream tag defines the conversion to be made from the input stream (received in a feed) to the output stream, which will be available to the users as a streaming service.

4. FFserver has a special status stream. This stream defines an HTTP page with the status of the FFServer. This is very useful for debug and management purposes.

```plaintext
# Global Configuration
Port 8090
BindAddress 0.0.0.0
MaxHTTPConnections 2000
MaxClients 1000
MaxBandwidth 1000

# Definition of the live feeds. Each live feed contains one video
# and/or audio sequence coming from an ffmpeg encoder or another
# ffserver. This sequence may be encoded simultaneously with several
# codecs at several resolutions.
<Feed feed1.ffm>
File /tmp/feed1.ffm
FileMaxSize 200K
ACL allow 127.0.0.1
</Feed>

# Definition of the streams. Each stream will be generated from the
# original audio and video stream. FFServer will send this stream when answering a
# request containing this filename.
<Stream test1.mpg>
Feed feed1.ffm
Format mpeg
AudioBitRate 32
AudioChannels 1
AudioSampleRate 44100
VideoBitRate 64
VideoBufferSize 40
VideoFrameRate 3
VideoSize 160x128
AudioCodec mp2
VideoCodec mpeg1video
</Stream>

<Server status stream>
Format status
ACL allow localhost
ACL allow 192.168.0.0 192.168.255.255
</Stream>

3.2.2 GStreamer

GStreamer is a framework for creating streaming media applications. It is possible to use streamer for just transcoding tasks as well as for streaming tasks. GStreamer uses a pipeline design, which is described below, and makes no extra overhead on top of the one introduced by the applied filter. This makes GStreamer a good framework for designing even high-end audio applications which puts high demands on latency.

GStreamer Pipelined Architecture

One of the the most obvious uses of GStreamer is using it to build a media player. For such purpose, GStreamer already includes a set of components for building a media player that can
support a very wide variety of formats, including MP3, Ogg Vorbis, MPEG1, MPEG2, AVI, Quick-time, MOV and so on. Currently, it also supports streaming over RTSP/RTP/UDP/TCP. Its main advantages are that the pluggable components can be mixed and matched in an arbitrary pipeline topology so that it is possible to write a full-fledged video or audio editing application.

The framework is based on plugins that will provide the various encoding formats and all other functionalities. The plugins can be linked and arranged in a pipeline, which defines the flow of the media. GStreamer pipelines can also be edited with a GUI editor and saved as a XML configuration file, so that pipeline libraries can be made with a minimum effort.

An example pipeline, illustrated in figure 3.5, can be used to broadcast an arbitrary media stream. In this example a raw media stream (can be audio or video) is passed to GStreamer. The second pipeline element has the task of encoding it. The third pipeline element prepares the encoded stream with the required headers of streaming protocol. The forth, and last, pipeline element sends the encoded streams with the headers to the desired host.

Hence, the GStreamer core function is to provide a framework for plugins, data flow and media type handling/negotiation. It also provides an API to write applications using the various plugins [31].

gst-launch

Gst-launch is a command line tool, distributed with GStreamer, to implement processing pipelines in a practical way. It uses the GStreamer framework and can be viewed as an interface for the wide side of possibilities of the pipeline design. It is also very useful for creating simple scripts for streaming and converting media formats. Example:

gst-launch audiotestsrc ! mulawenc ! rtppcmupay ! udpsink host=TARGET_PC_IP port=5555

The command above is an example of a gst-launch pipeline. In this instance of gst-launch there are four pipelines:

1. audiotestsrc generates an audio stream for debugging purposes. This first pipeline defines where the source stream is coming from.

2. mulawenc encodes the raw audio coming from the previous pipeline to a Mulaw encoding format which is also known as G.711.

3. rtppcmupay adds payload information to the previous pipeline for streaming through RTP.
3. Software Development Frameworks and Libraries

4. udpsink sends the stream from the previous pipeline (which is an G.711 audio stream encapsulated with the necessary payload information that defines an RTP stream) through an UDP transport channel.

3.2.3 Flumotion

Flumotion is a streaming software based on the GStreamer framework, which uses its encoding capabilities and adds more powerful streaming features, such as streaming using HTTP protocol. Flumotion aims to build an end-to-end software solution that integrates acquisition, encoding, multi-format transcoding and streaming of contents. It also allows the processing to be spread across multiple machines, so that the platform can scale to handle more viewers of more streams in more formats. All this is done using a modular approach. This means that the developer who is configuring a Flumotion Server can add and remove functionality modules in a modular way. The modular approach is a direct consequence of using gstreamer and its pipeline approach.

Flumotion server can have two distinct types of streaming services: Live Streaming and On Demand content. Live Streaming is the live acquisition of multimedia content, which is immediately streamed to remote clients. On Demand content can be viewed as a repository of previously stored multimedia files which can be streamed to remote users.

Flumotion Architecture

A Flumotion system consists of several parts. The most important components to be used in a Flumotion system architecture are:

Components  A single Flumotion system is usually denoted as a Planet. It contains several components working together. Some of these are Feed components, which handle task as receiving data, encoding it, and streaming it. This group of Feed components is called a Flow. This single architecture can be regarded as a flowing river on the surface of the Planet. Each Flow component outputs data that is taken as an input by the next component in the Flow, transforming the data step by step, exactly like the GStreamer pipelines.

Other components may perform extra tasks, such as restricting access to certain users, or asking users to pay for access to certain content (Bouncer components). This group of control components is called an Atmosphere. Again, it is part of the Planet.

The Flumotion architecture is better understood when looking at a diagram that shows all the components, flow and atmosphere. This is illustrated in figure 3.6 where it is shown an overview of Flumotion's architecture. Its main intervenients are:

- Feed Components: Feed components are arranged in a flow, connected to each other in sequence. Each component may be in only one flow and must be connected to another com-
3.2 Multimedia Streamer Frameworks

Figure 3.6: Flumotion Planet Architecture.

ponent in the same flow. Again, this set of restrictions are similar to those in the pipelines of GStreamer. There are three types of feed components:

- **Producer**: A producer only produces stream data, usually in a raw format, though sometimes it is already encoded. The stream data can be produced from an actual hardware device (webcam, FireWire camera, sound card, etc), by reading it from a file, by generating it in software (test signals), or by importing external streams from Flumotion servers or other servers. A feed can be simple or aggregated. For example an aggregated feed might produce both audio and video data.

- **Converter**: A converter converts stream data. It can also encode or decode a feed. It can combine feeds or feed components to make a new feed. It can modify the feed by changing the content, overlaying images over video streams, compressing the sound, etc.

- **Consumer**: A consumer element only consumes stream data. It might stream a feed to the network, making it available to the outside world, or it could capture a feed to disk.

- **Other Components**: The other components are part of the Atmosphere. They provide additional functionality to media flows and to the processing manager and are not directly involved in the creation or processing of the data stream. The only component which is part of the Atmosphere is:

  - **Bouncer**: A bouncer implements an authentication mechanism. It receives authentication requests from a component or manager and verifies that the requested action is allowed.

**Managers and Workers** A Flumotion system consists of a few server processes (daemons) working together. A Worker process creates child processes for the Components, while a Man-
3. Software Development Frameworks and Libraries

![Flumotion pipeline architecture](image)

Figure 3.7: Flumotion pipeline architecture.

The Flumotion administration process tells the Worker what to do. The Flumotion administration user interface connects to the Manager, which in turn controls the Workers, telling the components to start or to stop.

The components may be split across multiple machines, so there may be multiple Workers, usually one per machine. The entire system usually only has one Manager, on one of the machines. However, different arrangements, such as multiple workers and managers on one machine, are possible and may be useful in some circumstances in order to clearly separate different parts of the Flumotion system.

Figure 3.7 illustrates the Planet again, with specific components grouped into Workers on separate computers, and a Manager to manage the workers. After the manager process starts, it starts an internal Bouncer component used to subsequently authenticate the workers and the components. After that, it waits for incoming connections from workers. Then it can tell the workers to start their components. Those new components will also log in to the manager.[6]

3.3 Summary

This chapter presented the open source frameworks and software tools available for developing multimedia video conference applications.

It were described the PTLib, H.323 Plus and OPAL frameworks. PTLib is a supporting framework that is both used by the H.323 Plus and OPAL frameworks. PTLib ensures that these call signaling protocols can be used in multi-platform applications.

H.323 Plus is a framework that implements most of the H.323 standard. On the other way, OPAL focus on giving the developer a framework to implement call signaling protocols and use them. OPAL comes already with the SIP and H.323 protocols extended in the framework.

H.323 Plus is an obvious choice for applications that require deep functionalities of the H.323 standard. If the goal is to develop an application that can communicate with both the H.323 and SIP, the better choice is to use the OPAL framework.
4

Current Solutions

Contents

4.1 Hardware Solutions ................................................. 44
4.2 Software Solutions .................................................. 49
4.3 Summary: Hardware/Software Solutions Comparison ............. 56
4. Current Solutions

In this chapter, it will be presented some currently technology solutions for video conference, either by using specific hardware implementations or software implementations.

4.1 Hardware Solutions

Hardware solutions, also referred to as embedded systems for video conference, do not need any kind of personal computer since all the codification and communication is done by dedicated hardware boards. Usually, these kind of solutions have application oriented input and output ports. The Input ports should be used for video and audio sources, while the output ports should be connected to audio and video players. There is also another kind of input connectors (Ethernet, ISDN, etc) which add flexibility for a wide variety of different tasks.

4.1.1 Aethra AVC 8400

The Aethra AV 8400 [4] is a rack-mount hardware video conference system, in which the communication protocol relays on implementations of H.323 and H.320 standards. The multiple features offered by Aethra AVC 8400 include multiple network connectivity, simultaneous dual-stream video, multipoint conferencing and custom graphic user interface.

Network connectivity

Aethra AV 8400 has multiple network interfaces, which can be simultaneously used. These network interfaces have support for three types of networks: ISDN (BRI, PRI), Ethernet and V.35/Leased networks. The base configuration of this hardware include:

- 3 x BRI, with integrated channel aggregator;
- 2 x Port 10/100BASE-T full-duplex, with integrated switch Ethernet.

1ISDN BRI (Basic Rate Interface) is a standard ISDN service meant for residential and small scale business Internet connections.
2ISDN PRI (Primary Rate Interface) is a standard ISDN service designed to provide higher bandwidth than ISDN BRI.
4.1 Hardware Solutions

Optionally, other versions of this hardware may include:

- 6 x BRI, with integrated channel aggregator;
- 1 x PRI E1/T1, with integrated channel aggregator;
- 1 x G.703;
- 1 x V.35.

All network interfaces use RJ-45 plugs, except V.35 which uses 44pin Hi/Den plug. An example of the several inputs and outputs offered by this hardware is shown in figure 4.1.

Video/Audio

The set of video and audio capabilities offered by this hardware provide up to 8 video/audio inputs and 4 video/audio outputs. It also offers several types of input and output ports to connect different kinds of devices.

S-Video, Composite (BNC) and XGA are the video input types available in Aethra AV 8400 to transmit the captured video media stream. The video output ports enable the connection of different devices to reproduce the received video media stream. With these output ports, up to three monitors (using Composite and S-Video) can be connected. It also offers XGA output for connecting another type of video player device. This device also offers the option to control remote cameras using H.281 over ISDN and/or H.282/H.283 over IP Networks.

The audio inputs permit the connection of three different device types: 2 digital audio, 2 microphones and 3 line audio inputs. With digital audio, connected by a RJ-11 6/6 plug, it is possible to connect a microphone Pod with 360 coverage. Mic level enables the use of XLR audio sources. Line audio devices are connected through the RCA ports and can be used for most media devices used in everyday life. The audio outputs are offered through 4 audio line RCA plugs.

The supported codecs to encode/decode transmitted/received video stream are: H.261 [35], H.263++ [39], H.264 [40] and H.239 [37]. There are also several parameters on these codecs that can be configured. Video frame rate can be set either to 15 or 30 frames per second, using 56 kbps to 128 kbps and 168 kbps to 3 Mbps respectively. Video resolution can also be set to: 1024x768 (using XGA) 4CIF (704x576), FCIF (352x288) and QCIF (176x144) . The possible audio codecs in this hardware device consist of: G.711 (56 kbps), G.722 (48/56 kbps), G.722.1 (24/32 kbps) and G.728 (16 kbps). The audio codecs also have options for echo cancellation, adaptive post filtering, auto gain control and automatic noise suppression.

---

3The XLR connector is a style of electrical connector, primarily found on professional audio, video, and stage lighting equipment. They are most commonly associated with balanced audio interconnection.
4. Current Solutions

Figure 4.2: Tandberg 990 device.

MCU

This hardware can also be used for multipoint conferencing using the built-in Multipoint Control Unit (MCU). The MCU enables up to 7 simultaneously participants in the same call, using IP and ISDN networks by using Dial-In/Dial-Out capabilities, it enables participants to join conferences in progress. The initial list of video codecs supported by this system is reduced when the MCU mode is used. Only H.261 and H.263++ are available in this mode. To control the multi-conference, the H.243 protocol is used.

4.1.2 Tandberg 990/880/770

The Tandberg, illustrated in figure 4.2, hardware solution for video conference is a portable set-top unit already packed with a W.A.V.E (Wide Angle View) camera. Since it has built-in peripherals, all the external devices are a monitor, a microphone and speakers. This device can work both with H.323 and the SIP video conference signaling IP network protocols. Connecting to ISDN networks is also possible, by using the H.320 protocol. Tandberg device features include multi-conference (MultiSite), dual video streaming (DuoVideo), embedded security encryption, wireless LAN connectivity and IPv6 network support.

Network connectivity

The support for different types of networks makes Tandberg to include several different interfaces for connectivity purposes. These include: ISDN BRI, Ethernet, PC card slot (PCMCIA), V.35 and USB port in the following configuration:

- 4 x ISDN BRI (RJ-45);
- 1 x LAN/Ethernet (RJ-45) 10/100 Mbit;
- 1 x PC card slot (PCMCIA);
- 1 x V.35;

---

4 A Multipoint Control Unit (MCU) is a device commonly used to bridge videoconferencing connections.

5 Dial-In/Dial-Out features enables video conference participants to join and leave a conference dynamically without the need to setup the conference room each time a participant wants do join/leave.
4.1 Hardware Solutions

- 1 x USB.

The PC card slot and the USB are not directly involved in connectivity. The PC card slot is used to connect a Wireless LAN board, enabling the usage of this device without needing network cables. The USB interface is currently not in use.

Tandberg supports both and simultaneously IPv4 and IPv6 media and Net services. These services are: Telnet, Secure Shell (SSH), HTTP, Hypertext Transfer Protocol Secure (HTTPS), File Transfer Protocol (FTP), Simple Network Management Protocol (SNMP)\textsuperscript{6}, Domain Name System (DNS), Network Time Protocol (NTP), Dynamic Host Configuration Protocol (DHCP), H.323 and SIP Streaming.

Video/Audio

Even though the Tandberg 990/880/770 device has a built-in camera, it also has support for additional inputs and outputs both for audio and video: 4 video/audio inputs and 4 video/audio outputs. Almost all input and output ports are different, enabling the connection of many different types of peripherals. The next paragraphs presents further details about both the video and the audio features, as well as the supported set of codecs.

There are three types of video inputs: S-Video, to connect a video camera; RCA, to enable the connection of video cameras and VCR; DVI-I, to connect a personal computer. In what concerns the video outputs, monitors can be connected using either: S-Video, RCA or XGA output ports.

This device also offers connectivity for multiple audio peripherals, both for output and input. Audio input ports include two XLR connectors with a powered pre-amplifier, to connect high quality microphones. To connect RCA microphones or other devices with line out this device offers two RCA line level input ports.

The video codecs available on Tandberg line series are: H.261 [35], H.263 [39] and H.264 [40]. These video standards enable the hardware to receive and transmit a wide range of video formats: NTSC, PAL, VGA, SVGA, XGA, W-XGA, SXGA and HD720p, with resolutions as low as 400p (528x400) to wide resolutions as high as w720p (1280x720). It can also transfer still images in: CIF, SIF, 4CIF, VGA, SVGA and XGA formats. All these video formats work in two frame rates: 30 frames per second, at 168 kbps, or 60 frames per second, at 336 kbps. The audio codecs this device supports are the following: G.711 [32], G.722 [33], G.722.1 [36], G.728 [34] and 64/128bit MPEG4AAC-LD [15]. Allied to these codecs, this device features several functions to boost audio quality and to polish video conference experience, such as acoustic echo cancelers, automatic gain control, automatic noise reduction and active lip synchronization.

\textsuperscript{6}Simple Network Management Protocol (SNMP) is an internet standard protocol for managing devices on IP networks.
4. Current Solutions

![Polycom QDX 6000 device](image)

Figure 4.3: Polycom QDX 6000 device.

**Multipoint Conference**

the offered MCU, referred to as Tandberg MultiSite, enables the connection of up to 4 video and 3 audio sites in one big conference. MultiSite also enables various protocols (H.323, H.320, SIP, Telephony and VoIP) to co-exist in the same conference and talk to each other. Additionally, any site can receive dual stream video from each other site. To facilitate the entrance and departure of participants, the device offers Dial in and Dial out technology, enabling new participants to entry or leave a room without interrupting the multi-conference.

4.1.3 Polycom QDX 6000

The Polycom QDX 6000 device [23], illustrated in figure 4.3 aims to bring to the market a video conferencing product with emphasis on cost/quality relationship. This means that it does not necessary offers high quality characteristics in order to offer the best compromise in terms of price to its hardware competitors. This system was recently acquired by the Informatics Department of Instituto Superior Tecnico (IST). The main application of this equipment in IST is establish a seamless communication means between its two campi (Alameda and Tagus Park).

The Polycom QDX 6000 implements both SIP and H.323 video conference protocols, which enables it to communicate through an IP Network with virtually all other available video conference solutions. It supports dual streams and comes with a camera, two microphones and a remote control. The interface to manage the system is web based using SNMP.

**Network Connectivity**

As it was mentioned above, Polycom QDX 6000 uses SIP and H.323 signaling over IP Network to communicate with other video conference solutions. Therefore an RJ45 connector is present for connectivity purposes. Polycom QDX 6000 implements security mechanisms on both video
and audio streams, by encrypting each stream with the AES\(^7\) encryption algorithm. To lower the possibility of interrupted audio/video streams, Polycom developed the Polycom Lost Packet Recovery, which is also available on the QDX 6000. The Polycom Lost Packet Recovery aims to "provide smooth and uninterrupted conference experiences, even on congested networks" \(^23\).

**Video/Audio**

When purchased, Polycom QDX 6000 includes a high definition camera, two microphones and a remote control. The camera (Polycom EagleEye III Camera) offers up to 1080p resolutions, 12x optical zoom and an 72 degree field of view.

Polycom QDX 6000's video codecs include H.261, H.263 and H.264, with resolutions of 4CIF, CIF and QCIF. The video input ports are: 2x S-Video, one for Polycom EagleEye Camera; 2x Composite video and 1x VGA. The output ports include outputs for a main monitor and a secondary monitor. The main monitor is connected through a S-Video or a Composite Video (RCA) port, which are capable of outputting in 720p resolutions. The second monitor may be connected either through a VGA port, an S-Video or a Composite Video (RCA) port.

In what concerns audio, the codecs available in this equipment are (among others): G.719, G.722, G.711. It also includes some Polycom proprietary audio codecs. Polycom QDX 6000 is able to adjust the audio streams when the acquisition of audio is not being properly done, by applying algorithms for automatic gain control, automatic noise suppression, instant adaption echo cancellation and audio error concealment. The available ports for audio input are: 2x RJ9 for Polycom analog microphones and 2x Dual RCA. For audio output, Polycom offers 2x Dual RCA ports \(^23\).

### 4.2 Software Solutions

Software solutions assume that a personal computer with Internet connection will be used to run video conference applications. Software solutions (as hardware solutions) are commonly responsible for the communication and the encoding processes. Software solutions can also be responsible for just the communication protocols, being the codification process the responsibility of some kind of hardware codec. To use this type of solution the user must have a capable personal computer which should also have a reasonable number of input ports and/or peripherals to record media signals. This is also true for output peripherals, where personal computer must have at least one kind of interface for playing video and/or audio. While this may be seen as a limitation, it may actually become an advantage. If the application supports, the user can connect as many video/audio streams as he wants, not being dependent to the number of ports the hardware offers (contrasting to what happens with hardware solutions), as long as the computer supports it.

\(^7\)Advanced Encryption standard (AES) is a specification for the encryption of electronic data.
4. Current Solutions

4.2.1 VCON vPoint HD

vPoint HD [9], illustrated in figure 4.4, is a software-only client, running over MS Windows, which offers personal conference using either SIP, H.323 or H.320 signaling protocols. It can transmit and receive video, audio and data streams. vPoint HD incorporates H.329 protocol (HD DualStream™) for simultaneously sending and receiving video and data streams. Making use of VCON's SimulCast™ technology, it can also participate in multi-conferences by being the chair leader or a participant. Finally, it also includes recording and streaming capabilities, as well as the ability to connect across firewalls and NAT, being fully compliant with H.460.18/19 firewall/NAT traversal protocols.

Video and audio

By using the latest H.264 video standard, vPoint HD is able to send and receive high definition video. Users can participate in videoconferences at data rates ranging from 64Kbps to 4 Mbps, depending on the available bandwidth. The image quality provided by 4CIF and 720p resolutions offer sharp images in natural color and full motion. vPoint HD, the audio features include full duplex echo cancellation, automatic noise suppression and wide-band audio in low bit-rate calls.

VCON vPoint's supported video codec standards are H.261, H.263+/++ and H.264. These can achieve up to 15 frames per second (at 64kpbs to 128kpbs), up to 20 frames per second (when bandwidth is at 192kpbs) and finally up to 30 frames per second (when bandwidth is more than 256kpbs). It also supports several view modes for the incoming video. The user has the option to choose from mini screen, regular screen, full screen to 16:9 wide screen.

The supported set of audio codec standards include G.772.1, G.772.1, G.711, G.723, G.728, G.729 and AAC-LD, in a frequency range of 3.4KHz to 20KHz, thus offering a wide option of codecs to choose from, depending on the circumstances and network connection signal strength.
4.2 Software Solutions

4.2.2 Skype

Skype is a popular videophone software application that allows users to make voice calls over the Internet [28]. Calls to other users within the Skype service network are free, while calls to both PSTN and ISDN telephones and mobile phones can be made for a fee using a debit-based user account system. Skype has also become popular for its additional features, which include instant messaging, file transfer, and video conferencing. Skype uses a proprietary telephony network protocol, called Skype Protocol. The protocol has not been made publicly available by Skype and official applications using the protocol are closed-source. Part of the Skype technology relies on the Global Index P2P protocol. The main difference between Skype and standard VoIP clients is that Skype operates on a peer-to-peer model, rather than the more usual client-server model (SIP and H.323 are also peer to peer, but their implementation generally requires a registration with a server, as does Skype).

Even though Skype has the feature of screen sharing, it cannot be seen as a video conference solution because it cannot transmit two video sources at the same time (e.g. screen sharing and webcam).

Skype has a variety of clients for different platforms: Linux, Linux-based Maemo, Symbian S60, Mac OS X (Intel and PPC), iOS (iPhone and iPod Touch), Android, Microsoft Windows (2000, XP, Vista, 7, Mobile).

4.2.3 Ekiga

Ekiga, shown on figure 4.5, is an open source videophone and instant messenger application over the internet. It was implemented using OPAL (and therefore PTLib) framework, so it is multi-platform compliant and can communicate with most VoIP service providers as it implements SIP and H.323 standards. Ekiga software was formerly known as GnomeMeeting.

Ekiga does not impose a specific service provider to be used with. Instead, the user can register to any H.323 and/or SIP service provider it wants. This gives the user the possibility of using service providers features like: text messaging, calling cell phones, voice mail, Information Processing in Sensor Networks (IPSN) number, etc.

Ekiga also offers standard telephony features support like call hold, call transfer, call forwarding and DTMF giving the user the freedom to manage calls the way he wants to. DTMF support enables Ekiga to call automatic answer machines and generate DTMF to control the flow of the call (eg “press 1 to proceed”).

Ekiga’s audio and video codec management permits the disabling and re-ordering of codecs. This enables the advanced user to choose the specific codecs it wants to use, depending on the bandwidth available. Choosing an ordered list of codecs forces Ekiga to try the user preferred
4. Current Solutions

![Ekiga Software Screenshot](image)

Figure 4.5: Screenshot of Ekiga software.

codecs first, while still enabling successful calls (unless the enabled codec list is too narrow, which may fail if the destination call does not have the codecs chosen by the user).

Another advanced function available in this software: H.245 tunneling, early H.245, and Fast Start.

### 4.2.4 Asterisk

Asterisk [7] is an open-source communications platform, which can run on Unix, OpenBSD, FreeBSD and Mac OS X operating systems, that converts a personal computer into a voice communication server. Asterisk enables the deployment of several telephony applications and services, including: IP Private Branch Exchange (PBX), VoIP gateways, call center, Automatic Call Distributor (ACD) and Interactive Voice Response (IVR) systems.

Even though Asterisk is an application, it can be seen as a “toolbox” or “development platform” for building any kind of communication solution. Asterisk offers a wide range of built-in features:

---

8H.245 Tunneling is the encapsulation of H.245 messages within H.225/Q.931 messages (H.245 Tunneling). If a firewall and H.245 Tunneling are active, there is one less TCP port that is needed to allow for incoming connections.

9Early H.245 enables H.245 early in the setup and permits to achieve faster call initiation.

10Fast Connect is a new method of call setup that bypasses some usual steps in order to make it faster. In addition to the speed improvement, Fast Connect allows the media channels to be operational before the CONNECT message is sent, which is a requirement for certain billing procedures.

11PBX stands for Private Branch Exchange, which is a private telephone network used within a company. The users of the PBX phone system share a number of outside lines for making external phone calls.

12Automated Call Distribution is a device or system that distributes incoming calls to a specific group of terminals that agents use.

13Interactive Voice Response is a technology that allows a computer to interact with humans through the use of voice and DTMF tones input via keypad.
components. In fact, since it is open-source there is no limit on what it can be made to do. The combination of these components enables the integrator or developer to quickly create and deploy voice-enabled applications. Some of these components are:

- Drivers for various VoIP protocols;
- Drivers for PSTN interface cards and devices;
- Routing and call handling for incoming calls;
- Outbound call generation and routing;
- Call detail recording for accounting and billing;
- Transcoding (conversion from one media format to another);
- Protocol conversion (conversion from one protocol to another);
- Single and multi-party call bridging;
- Speech synthesis (aka “text-to-speech”) in various languages and dialects using third party engines;
- Speech recognition in various languages, using third party recognition engines.

Asterisk supports a wide range of protocols for handling and transmitting voice over the most common communication standards, including H.323, SIP, Google Talk, Media Gateway Control Protocol (MGCP), and Skinny Client Control Protocol (SCCP). By using the Inter-Asterisk exchange (IAX) Voice over IP protocol, Asterisk merges voice and data traffic seamlessly across disparate networks. The use of Packet Voice allows Asterisk to also send data (such as URL information and images) in-line with voice traffic, allowing an advanced integration of information. Asterisk also features a wide-range of codecs it can use for transcoding services, some of them are: G.711, G.722, GSM and Speex.

4.2.5 Adobe Connect

Adobe Connect [14], illustrated in figure 4.6 is a proprietary web conferencing solution for Web Meetings, eLearning and Webinars. It is available to run in many different devices such as a PC, Tablet or Smartphone, due to its web based interface. In fact, since Adobe Connect's web interface is based on Adobe Flash technology, this makes it easier for users to join sessions, since there is little to none setup required. For mobile devices, Adobe developed Blackberry, Android and iOS applications, so that Adobe Connect can be run virtually on every device.

The Adobe Connect conferencing solution goes beyond video and audio. Users can also collaborate using rich and dynamic content such as white boards, applications, quizzes, etc, as described below:
4. Current Solutions

![Adobe Connect screenshot](image)

Figure 4.6: Adobe Connect screenshot.

- **Web Meetings**: The Web Meetings module allows users to collaborate more efficiently than by just using audio and video. It also allows users to share real-time content, such as whiteboards, screen sharing, notes, presentation slides and other collaboration tools [2].

- **eLearning**: This module enables managers and content providers to create on-demand courses, conduct interactive virtual classes and manage training programs [1].

- **Webinars**: Webinars can be seen as virtual seminars where speakers present a subject to an online audience. Adobe Connect also provides this feature to its users. Some of the offered features are: pools, question and answer, hand raising, notes, whiteboards, and document sharing such as pictures, videos and documents [3].

Since Adobe Connect is a pure software solution, its performance is highly dependent on the hardware chosen to the servers and clients. It uses a proprietary communication protocol for audio and video. Given that, it also uses other type of rich content so that it is nearly impossible to have the same amount of interactivity with other types of video conference solutions. Despite that, it is possible to use Flash Media Gateway\(^{14}\) to provide SIP communication with other video conference solutions.

Adobe Connect conference also allows multiple video feeds from the users. This enables each user attending the video conference to see the other users.

### 4.2.6 Microsoft Lync

Microsoft Lync, illustrated in figure[4.7] is an instant messaging client with support for video/audio conferencing and collaboration tools. Its main advantage over its competitors is the unified approach Microsoft took in developing Lync. It can be linked with other Microsoft products such as

---

\(^{14}\)Flash Media Gateway is a new real-time server platform that enables Adobe applications to connect with traditional communication devices via SIP.
4.2 Software Solutions

Word, Sharepoint and Outlook. This makes the users feel immersed in a collaboration/communication bubble. Lync acts as the communication and collaboration brain in a Microsoft deployment environment [19][20].

Microsoft Lync can be accessed by various medium. These include: Microsoft Windows, web access, mobile devices and even Mac OSX. This does not mean that every feature is available in every device; features go as limited as only being able to use instant messaging.

Multipoint Conferencing

Microsoft Lync supports both audio and video multipoint conferencing. While the audio conferencing is done in a similar way just like its competitors, the video conferencing is not done in a traditional way. The common solution for multipoint video conferencing (more than two participants) is to display all video streams from every participant. Microsoft Lync developed a different way to achieve this. There is only one active video at once, which is defined by the active speaker. This means that only the participant who is talking is being displayed in the video.

Signaling Protocols and Audio/Video Codecs

Microsoft Lync uses SIP as its signaling protocol. This means that it can communicate with all audio/video conference solutions that also use SIP.

The audio codecs used by Lync are: RTAudio\textsuperscript{15} G.711, G.722 and Siren. The video codecs provided by Lync are RTVideo\textsuperscript{16} and H.263.

\textsuperscript{15}RTAudio is a proprietary audio codec developed by Microsoft to be used in real-time scenarios.  
\textsuperscript{16}RTVideo is a proprietary video codec developed by Microsoft to be used in real-time conferencing.
4. Current Solutions

By using a gateway, Microsoft Lync can be extended to provide access to H.323 protocol and H.261 and H.264 video codecs.

4.3 Summary: Hardware/Software Solutions Comparison

To conclude the presentation of the several solutions available in the market that were studied in this chapter, a table (illustrated in figure 4.8) will be presented for a simple comparison between all of them. The features that were compared are those that are considered as the most important for this research. Among them, a particularly importance was given to its compatibility with other solutions (if it uses H.323 and/or SIP protocols), support for dual streaming, support for more than two participants and its market availability (free/paid).

The conclusions that can be taken from this table can be broken down into the following:

- Most solutions make their top priority the usage of proven communication protocols. The choice of H.323 and SIP is also explained because they are communication standards. As a consequence, it is considered an advantage to use them when a new video conference solution is being developed because this assures (assuming that standard audio/video codecs are also being used) its integration with the other solutions. Only Adobe, Microsoft and Skype, opted to implement their video conference applications with their own proprietary

```
<table>
<thead>
<tr>
<th>Solutions</th>
<th>Features</th>
<th>SIP</th>
<th>H.323</th>
<th>Multiple Video Sources</th>
<th>Multipoint Control Unit (video)</th>
<th>Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aethra AVC 8400</td>
<td></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Tandberg 990/880/770</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Polycom QDX 6000</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Software</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCON vPointHD</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓*</td>
</tr>
<tr>
<td>Skype</td>
<td></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>Ekiga</td>
<td></td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Asterisk</td>
<td></td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Adobe Connect</td>
<td></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>⬲</td>
</tr>
<tr>
<td>Microsoft Lync</td>
<td></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

* Skype offers support for a three way (or more) video conference only in paid version.
** H.323 compatibility can only be achieved using a proprietary gateway and additional tweaking.
```

Figure 4.8: Current solutions comparison table.
communication protocols. This can also be explained by the wide range features they provide beyond simple video and audio calls.

- None of those solutions provide simultaneous multiple video sources for free. Skype comes close to this, in its free option. It can use, for example, a screen sharing the video source but unfortunately it cannot stream a webcam and the screen at the same time, so it cannot be catalogued as dual video stream.

- "Multiple video streams in a call" and "more than two participants in a video call" features come attached in the set of features each one of the solutions offers. If one is offered, the other one is also offered. This means that either a solutions implements both features or none. It can also be derived from this that the dual streaming (for a presentation session) feature is not a priority when it comes to developing a new video conference solution. Multiple video sources are only implemented as a requirement for a video call with more than two participants.

- There are not many free solutions that focus on video conference. As we can see only three solutions are free to use in this set.

- Perhaps the most important conclusion that it can be taken from this table is that currently there is not any free solution that can be used for a presentation session and properly satisfy its requirements (dual video stream being the most important requirement).
4. Current Solutions
Proposed Architecture
This chapter will focus on the developed solution to solve the problems that were earlier defined. It will be explained its architecture, what are the used technologies and used frameworks and why they were chosen.

5.1 System Architecture

Before starting the presentation of the solution’s architecture, it will be introduced the main modules and their interactions in an abstract way. Figure 5.1 illustrates an overview of the global system architecture of the proposed system. Since there will be a speaker in one side, and an audience in the other side, there will be two possible configurations of an instance of the system. The speaker side will feature a webcam to capture the speaker, a video grabber to capture the supporting media (i.e. presentation slides) and a microphone to capture the speaker’s voice. The speaker’s side will also feature a video projector to display the audience, and audio speakers to also play the audience questions.

In the audience side, there will be only one webcam and one microphone to correspondingly capture the audience’s image and their questions. The audience also features the speakers for reproducing the speaker’s voice and the video projector, this time, for playing both the speaker’s video and the supporting media video.
5.1 System Architecture

The architecture comprises three distinct modules: a Call Manager, a Multimedia Streamer and an Interface. They interact in different ways and most of the communication has the specific purpose of transferring multimedia streams. In the next paragraphs, each of these modules and the interaction between them will be explain in more detail, with help from figure 5.2.

The Call Manager module is responsible for all the communications with the remote clients, such as the acquisition and dispatch of video and audio, the capabilities management and all other aspects of call management. As an example, it is the Call Manager’s responsibility to initiate the process of answering and establishing a call. This process implies the acquisition of video signal from a webcam or a screen capture device and the audio signal from a microphone. It is also in the Call Manager that negotiates what will be the codecs used in the call as well as what video/audio streams are to be sent and received.

The Call Manager communicates with both the Multimedia Streamer and Interface modules. The communication with the Multimedia Streamer (represented by the connection number 1) is exclusively used for the Call Manager to transfer the multimedia streams it receives from a remote client to the Multimedia Streamer. Therefore, connection number 1 represents a set of multimedia streams.

The Multimedia Streamer module serves as a bridge between the Call Manager and the Interface. Its only job is to stream the multimedia streams it receives from the Call Manager to the Interface module. It implements a streaming standard in order to successfully stream without hassles and so that the Interface can present the streams as smoothly as possible.

It is important to note that connection number 2 is not similar to connection number 1. The difference is that connection number 2 has the streams in a format that the Interface can understand and present to the user. It would not be possible to simply bypass the Multimedia Streamer module and use only the Call Manager and the Interface modules because this would cause problems that will be described in later sections.

Lastly, the Interface module’s work is, as the name suggests, the visible part of the whole solution’s architecture. The Interface presents the multimedia streams to the user, as well as it provides the user with the power to interact with the application. This interaction includes the change of video definitions, the acceptance of call ad the start of calls.

The last connection to be described is connection number 3. This connection is a two way connection. From the Call Manager to the Interface, this connection represents the status updates imposed by the Call Manager to the Interface (e.g. when a new call needs to be accepted, the Interface must change in order for the user to check its details and accept it). The other way of the connection, i.e., from the Interface to the Call Manager, it allows the actualization of general application settings such as the video and audio sources to be used.

This section provided a brief description and an introductory point to the modules that compose
5. Proposed Architecture

The next step towards describing the architecture of the solution is to introduce the technologies that were chosen to implement each of the modules. Since the final implementation is dependent of the adopted frameworks and technologies, this section will present all the choices that were made in terms of the 3rd party software that was included in the solution to build the final application.

5.2 Software Frameworks

In figure 5.3 it is illustrated the crossover from the abstract high level architecture to its actual realization using frameworks and software as modules. As it is illustrated, the main module (Call Manager) is implemented using the OPAL Framework. Flumotion is the streaming server chosen to act as the streaming module. The Interface module is implemented using an HTML webpage.
5.2 Software Frameworks

solution. The integration between these three pieces of technology is the main challenge of this work and will be extensively explained in a later section. In this section it will be explained why these technologies were the chosen ones, among all the technologies that were investigated and analyzed.

5.2.1 Call Manager

The Call Manager module is the most complex module and it is where the core work is done. As it was described above, this module is responsible for the call management. This means that its responsibility comprises all aspects of network handling, capabilities negotiation, acquisition of media and many more. Due to its complexity it is beyond this work to implement a new protocol or an established ISO standard from scratch. To accommodate this module's job a framework that implements the whole call signaling and standard management is best option. In figure 5.4 it is represented a table with the open source frameworks that implement such standards/protocols.

The features that were chosen for this comparison are the ones that fit the best in what is expected from them in the solution to be developed. These features were chosen because:

- **H.323 and SIP**: It is important that the chosen framework implements the most common call signaling standards, which are the H.323 and SIP. Since the objective of this solution includes the possibility to communicate with most currently available applications for videoconference, and since most of those solutions adopt SIP and/or H.323, it is also mandatory that this solution also support these protocols.

- **Extensibility to other protocols**: Again, since one of the main objectives of this solution is the ability to communicate with a wide range of different solutions, a framework that enables the addition of other call signaling protocols is very welcome. It should be noted that this is not an eliminatory feature. Nonetheless, support of this feature is recommended.

- **Active Development**: A framework that is still being developed and maintained is simultaneously good and bad. It is good because it evidences that developers are still interested and motivated in the framework. It is bad because it also shows that the framework is not still completely developed. It is okay if the framework is not completely developed as long as it offers stable versions that implement the core functions of the standards. Moreover, with open source frameworks there is always the chance of appearing bugs with any update of either the operating systems or any dependent packages that are used by the framework. Consequently, it is really important that developers are also updating the framework, in order to make it compatible with most operating systems, hardware and software packages.

- **Community Support**: For a developer to use a framework there are some steps that need to be done before the framework becomes useful. The main ones are: installation, learning and deployment. In all these steps there are always many problems that can happen.
5. Proposed Architecture

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>Features</th>
<th>SIP</th>
<th>H.323</th>
<th>Extensibility to other protocols</th>
<th>Still Active?</th>
<th>Community Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenH323</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>H.323 Plus</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OPAL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Figure 5.4: Comparison table between the currently available call signaling frameworks.

Hence, it is very useful to have a knowledge base to fall back when a problem appears. This knowledge base is often provided by a good and active community that can personally support the developer in specific problems. This can come from an already established platform with questions from other developers such as a mailing list or a forum.

By considering all the above observations, the chosen framework was OPAL. As it can be seen in figure 5.4, OPAL fulfills all the requirements that were imposed when choosing a framework to implement the Call Manager module. Nevertheless, OPAL is not perfect and has some flaws, specially when trying to learn how to use it. The learning of OPAL needs to be done by digging its code and checking some applications and examples that were done with it. Ekiga was also developed using OPAL and its source code was read as a learning process.

It should be noted that OPAL does not only implement H.323 and SIP but tries to be open and flexible enough to implement other call signaling protocols. This extension is done by extending the core classes of OPAL, so that no source code needs to be modified inside OPAL. Finally, it is worth noting that OPAL does not fully implement H.323 and SIP. Neither it is their objective to do so. Despite that, the implementation for H.323 and SIP available in current versions are good enough to be used in the solution to be developed.

In latter sections it will be discussed how OPAL was used in the solution in order to integrate with the other modules: Interface and Multimedia Streamer. It will be also described in detail how the Call Manager module was developed.

5.2.2 User Interface

One of the main objectives of the proposed solution is to be multi platform. Considering this goal, the best and simpler way to achieve this is by using an HTML interface. HTML is a common technology that is available in most Operating Systems by means of a browser. It is also very easy to modify an HTML webpage for extending its functionality. Another alternative choice for building an interface is to use an SDK Tool Kit. Unfortunately, there is a problem when using an SDK Tool Kit for building interfaces:
5.2 Software Frameworks

![Comparison table between multimedia containers.](image)

Figure 5.5: Comparison table between multimedia containers.

1. With Operating System updates, the Interface may become obsolete and more maintenance is necessary in order to keep it working.

**Multimedia Container Formats**

The HTML interface to be implemented should be divided into two components: the presented text and the multimedia streams to be played. The multimedia streams are presented in multi-media containers. It is also the developer responsibility to choose what container to use in the interface because it may have a critical impact in the implementation of other modules. [43].

Figure 5.5 presents a table with a comparison between the most used multimedia containers. For this solution, all the major well know containers can be an option. The most important feature concerns the need of a plugin played in a browser. Unfortunately, most multimedia containers require a plugin to be installed in the browser in order to be played. The only container that does not need it is WebM. Hence, it is container adopted.

There are other aspects that are also important when choosing a multimedia container for real-time streaming. The chosen container must be encoded by a software tool. Thus, the adopted set of software tools must support the container. Moreover, the encoding speed is also relevant because the streaming should not stop because of any inability to encode at the desired speed.

**5.2.3 Multimedia Streamer**

The last module requires a more detailed definition in terms of technology is the Multimedia Streamer module. This module is responsible for gathering each media stream from the Call Manager module and sending the correspondent WebM containers over HTTP to the Interface Module. The most obvious features required from the multimedia streamer software should be the ability to support WebM encoding and stream using the HTTP protocol.
5. Proposed Architecture

<table>
<thead>
<tr>
<th>Streaming Server</th>
<th>WebM Support</th>
<th>HTTP Streaming</th>
<th>Multiple Multimedia</th>
<th>Live Streaming</th>
<th>Command Line Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>GStreamer</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FFServer</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Flumotion</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Figure 5.6: Comparison table between multimedia streaming servers.

Figure 5.6 represents a table that crosses multimedia streamers with the most important features. These features are:

- **WebM Support**: WebM container support is crucial since it was earlier defined the adoption of WebM in the Interface module.

- **HTTP Streaming**: There are several protocols for multimedia streaming as it was seen in a previous section. They are: RTP, RTSP, HTTP and others. Not all streaming servers implement all these protocols. In this solution the streaming protocol to be used is HTTP. That is why this feature is so important and was added in the table.

- **Multiple Media**: Multiple Media feature describes the ability of a streaming server to send more than one multimedia stream at the same time. Since the final solution will have two video streams and one audio stream (video and audio from the speaker and video from the presentation slides), the streaming server must be able to support two instances of itself. Among the considered servers, all of them are able to successfully do this task: GStreamer, by using two gst-launch instances; FFServer and Flumotion, by using a single instance to stream multiple multimedia streams.

- **Live Streaming**: Live streaming comes in two flavors: it can be a live streaming from a pre-recorded file, which is in fact called on-demand streaming, and literally live streaming from a stream that is being generated in real time. The last definition of live streaming is the one that is required for this solution. All of the covered servers can do live streaming.

- **Command Line Support**: Since the multimedia streaming server needs to be dynamically changed and without the user intervention, the server needs to be fully controllable from other software in an automated way. The way to do this is by using command line commands. Thus, the server needs to have a command line interface tool to define and describe the streams that will be streamed over HTTP. Again, all of the analyzed servers offer a command line interface.
Having these features in mind and the comparison table as guidelines, the choice was Flumotion but only because GStreams and FFServer did not performed as expected (as a lightweight server would have been preferred).

GStreamer does not support HTTP streaming and it was discarded since the start. FFServer, on the other hand, was a strong candidate to occupy the streaming server module. FFServer seemed to have a lightweight architecture and a simple way to work with it. Unfortunately, after many efforts to get it to work in the developing environment it was realized that FFServer could not stream WebM or any other container format to a browser. FFServer lacked support from the community and did not seem to be mature enough to be used in an application. Flumotion also presented some problems as it will be described in later sections. Nevertheless, it is able to successfully stream WebM multimedia streams to a browser, which is the ultimate goal. Moreover, the provided command line tools offers an automated alternative interface to launch the server with the right parameterizations.

5.3 System description and framework integration

In this section it will be described in better detail how the three described modules communicate with each other in order to implement the final solution (see figure 5.7). In past sections, the three modules were introduced and their responsibilities were also defined. In this section, however, the main objective is to describe the connections between them, as it is shown in figure 5.2. In the next sub sections, each module will be individually described in better detail.

5.3.1 Opal and Flumotion Integration

The Opal call manager and the Flumotion stream server modules are the first in the chain between the acquisition of remote video and the presentation of the composite multimedia streams in the interface. As it can be seen in figure 5.7 OPAL sends not only the singular media streams, but it also starts the Flumotion Server through the generation of the configuration XML files.

There are two notes that should made from figure 5.7 regarding the Opal - Flumotion integration:

- OPAL does not literally send two XML configuration files to Flumotion. The implementation decision was for OPAL call manager module to launch a Flumotion instance using the configuration files that it generated having into account the media stream properties, such as: framerate, frame size, audio channels, etc.

- Opal does not send media streams to Flumotion in a direct way. The implementation that was done in the OPAL Call Manager module creates a named pipe for each media stream
5. Proposed Architecture

Figure 5.7: Detailed solution’s modules interaction
(in this case two raw video streams from the speaker and the supporting media and one raw audio stream from the speaker) that is then read by Flumotion.

In figure 5.8 it is illustrated the instantiation process used by OPAL to launch Flumotion Server. The first comment that it should be done is that the Flumotion module will be, in fact, implemented using two Flumotion instances. This has to do with practical problems that appeared when trying to stream two multimedia containers in one single instance of Flumotion.

There are several steps that have to be handled by OPAL in order to launch Flumotion instances, as it is shown in figure 5.8. These steps are:

1. **Generation of the named pipes**: The first step in initializing the Flumotion’s instances, is creating three named pipes for: the speaker’s video, the speaker’s voice and for the supporting media.

2. **Initialization of Flumotion 1**: The initialization of Flumotion 1, which will be used for the speaker or audience’s video and voice, comprise several inner steps:

   (a) **Generation of the server configuration file**: The server configuration file will contain the definition of the media streams that will be used for the speaker or the audience’s video and voice. It will also contain the port number for the internal communication between the server and its workers. Finally, it will also be defined the port number for the HTTP streaming of the WebM multimedia container.
5. Proposed Architecture

(b) **Generation of the worker configuration file**: The worker configuration file will only contain the port number that is being used by the streamer.

(c) **Launch of Flumotion 1**: After the configuration files are created, the manager launches Flumotion through a command line, which will take as the input the two configuration files.

3. **Initialization of Flumotion 2**: The initialization of Flumotion 2 is similar from the initialization of Flumotion 1. The difference is that Flumotion 2 only needs to stream the supporting media video, because the supporting media does not have an associated audio media stream. Thus, its configuration file only features the definition of one video stream as its input.

When both Flumotion instances are launched, the named pipes have a reader that are waiting for data be written to. Flumotion stays in a waiting state, waiting for data to be written to the named pipes.

4. **A new call is started**: When a new call is started, OPAL manages all the specific details that will be described in later sections in order to negotiate and acquire the relevant media streams to send to Flumotion.

5. **Start writing data to the named pipes**: When the media streams are created, OPAL starts writing media data to the named pipes. Once Flumotion starts reading data from the named pipes, it starts encoding and streaming them to the web interface.

It is now clear how the OPAL call manager module and Flumotion streaming server cooperate. This indirect way of creating Flumotion instances and providing the RAW media streams are also an advantage for later tweaking. This means that if by any reasons a developer wanted to change the Flumotion module to an FFServer module this could be done in a straightforward way.

To summarize, for the integration between OPAL and Flumotion it was needed to implement the:

- Creation of the named pipes;
- Generation of the Flumotion XML configuration files;
- Launching of the Flumotion instances.
- Forwarding of the received media streams to the named pipes.

5.3.2 **OPAL and Interface Integration**

The integration between OPAL and the HTML interface is done using an HTTP server embedded in the Call Manager. This can be seen in figure 5.9. The developed HTTP Server is responsible for updating the interface, as well as dealing with the requests from the browser interface module. The HTTP Server is an important submodule of the whole architecture and it can
be seen as middle software part between the Call Manager and the Interface. All information that is passed between these modules is passed through this server.

As it was discussed before, the user interface is used both to play the multimedia streams to the user, as well as to change some definitions of the system and impose commands on the call manager. Figure 5.10 illustrates some example of actions that can be done within the interface that are dependent on its communication with the call manager. Again, this figure illustrates how the HTTP Server is the communication door between the call manager and the interface. To summarize, the Interface’s job is to:

- Show context information when a call is occurring: when a call takes place it is required the actualization of the interface (from idle to on call).

- Change video conference definitions: when the interface is on an idle state, the user can change specific application definitions. These definitions include changing webcams and audio input sources, frame rate, username, etc. To perform this, the interface must communicate with the call manager (which is the responsible module for storing definitions) in order to specify which definition is to be changed and what is its new value.

- Accept Calls from remote client: when a new call is being done from a remote client to the call manager, this module needs to inform the interface of a new call. To do this, the interface must be continuously asking the call manager for new information. When the interface is aware of some new information, it changes its interface in order for the user to accept or reject the remote call.

- Start calls to other clients: to start a call, the user supplies the interface with the remote address to where he wants to establish a call. It is the interface’s responsibility to transfer this information to the call manager and then change its interface when the remote client accepts or rejects the call.

5.3.3 Flumotion and Interface Integration

This is the last and most simple integration procedure that takes place in the proposed architecture. The integration between Flumotion and the interface is virtually almost transparent to the
5. Proposed Architecture

![Proposed Architecture Diagram]

Figure 5.10: Communication flow between call manager module and interface module.

developer. Since Flumotion streaming server sends the WebM streams through HTTP, all that
has to be done is to point the HTML5 video tag to the url that Flumotion generates.

In figure 5.11 it is illustrated a setup where Flumotion sends two WebM multimedia streams
over HTTP. One WebM stream is a container with both video and audio of the speaker/audience.
The other stream is only video and typically corresponds to the presentation slides (or supporting
media).

Hence, the interface module has two HTML5 video tags that acquire the video streams being
sent by Flumotion. One HTML5 video tag container will play the speaker’s video and voice. The
other HTML5 video tag container will play the supporting media.

The whole process, represented in figure 5.11, is internally controlled by Flumotion’s HTTP
server and by HTML5 video tag. They negotiate the streaming of the multimedia streams through
the HTTP protocol. Flumotion, in conjunction with its internal buffers, then streams the multimedia
streams to the HTML5 video that is part of the web interface.

5.4 Summary

In this chapter it was described the proposed architecture for the problems identified in chapter
1.

The proposed architecture includes three main modules: the call manager module, the stream-
ing server module and the interface module. The call manager module is implemented by an
OPAL application; the streaming server is implemented by the Flumotion streaming server and
5.4 Summary

The Interface module will be implemented using HTML.

It was also described in great detail how the three modules interact through the use of an HTTP Server, named pipes and streaming protocols.

Figure 5.11: Integration between Flumotion and the Interface Module.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 OPAL Call Manager Architecture</td>
<td>75</td>
</tr>
<tr>
<td>6.2 Flumotion Stream Server Configuration</td>
<td>81</td>
</tr>
<tr>
<td>6.3 HTML Interface</td>
<td>83</td>
</tr>
<tr>
<td>6.4 Summary</td>
<td>84</td>
</tr>
</tbody>
</table>
This chapter will be present how the proposed architecture was implemented and what choices were made over the course of the development phase.

6.1 OPAL Call Manager Architecture

In this section it will be described the implementation of the Call Manager, as well as the considered design options that were placed when developing the final solution.

6.1.1 Call Manager Architecture

The Call Manager module was implemented by an OPAL application. When OPAL was introduced in chapter 3, the main classes that compose OPAL were also described. In this section, it will be described which classes were redefined and extended and how they were implemented, in order to develop the final solution.

In figure 6.1 it is illustrated the set of involved classes, when a call is taking place. What is important to refer about this figure is the existence of an hierarchy of class instances at each moment of the established call. The manager controls the endpoints and informs each one that a new call can be answered.

The endpoints’s job are to be aware of outside information. In the case of the H.323/SIP endpoint, its job is to be listening for remote calls, as well as to send the media streams from inside OPAL to the remote client. The local endpoint is used to communicate with the user and with the media inputs, such as a webcam or a microphone. The Call object represents the call and it is responsible for managing both an H.323/SIP Connection and Local Connection. The Local Connection is responsible for transporting the local media streams (that acquire the audio and video input sources) to the H.323/SIP Connection, as well as, to receive the media streams from the H.323/SIP Connection. Each Connection has associated MediaStreams, that represent the type of information that is being passed from one connection to the other.

The MediaStreams transport information from one side to the other. These sides vary depending on which MediaStream is being used. In the example of figure 6.1 it is shown that the LocalConnection’s job is to acquire media streams from hardware (mic, webcam) and to transport it to the remote client. It is also its job to acquire the remote media from the H.323/SIP Endpoint and export it to a named piped where Flumotion will be connected to.

It is important to note that the instances for H.323 and SIP in figure 6.1 are being seen as one single instance. This is not the case in reality and it was presented this way to simplify the diagram.
Manager’s Implementation  The Manager is a singleton object (only has one instance) in the architecture and is the brain of the whole application. It orchestrates what happens when an instance is created, as well as the initialization of every object and the callbacks from every class. To summarize, the Manager class is responsible for:

- Initialize H.323, SIP and Local Endpoints.
- Initialize listeners for the H.323 and SIP: this task includes checking if the desirable ports are not being used by other application. In practice, it increments the port number until there is an open port available.
- Define the manager routing table: OPAL needs to know where to redirect both the received packets from H.323 and SIP endpoints and the received commands from the user (local endpoint). Therefore, the routing table should redirect the H.323 and SIP packets that were received from the respective endpoints to the local endpoint, and to transfer the local endpoint packets to both H.323 and SIP endpoints.
- Set the available capabilities list: before the initialization of the manager is finished, it has to setup the supported codecs and media formats to use when a call is initiated.
- Initialize the HTTP Server: the final step in the initialization of the Manager is to start the HTTP Server that will communicate with the user interface.
Local Endpoint's Implementation The local endpoint plays an important role in the conceived architecture. Since its role is to interface with the user, it is important to describe how it was implemented. The local endpoint has also the task of selecting the audio and video devices to grab the media streams from, according to the settings. Finally, it also has to redirect the streams received from the H.323 and/or SIP endpoints to named pipes. In the following paragraphs, it will be presented a brief description of those tasks:

Starting and accepting a call The HTML interface has the task of presenting and grabbing information from the user. It has two important tasks: enable the initiation of a call, and signaling a new call to the user (from a remote client). For the completion of both tasks, the HTML Interface needs to communicate with the Local endpoint. This is done by means of the HTTP Server, which is embedded in the Call Manager module. In case there is a call waiting to be answered when the HTML page refreshes, the HTTP Server returns a ringing page so that the user can accept or reject the call.

Grabbing the video/audio media streams Another task of the local endpoint is to define the devices that should be used to grab the video/audio streams. This is done when the Local Connection creates a new source stream. At this time, it should identify the device that will be used to grab the media stream from the application settings and create the appropriate media stream: `OpalAudioMediaStream` for audio streams and `OpalVideoMediaStream` for video streams.

Writing to the named pipes The communication between the Call Manager and the Streaming Module (Flumotion) is based on named pipes that transport the raw media streams. The creation and management of those named pipes are the responsibility of the Local Connection object. Just like the grabbing process, at the time the Local Connection creates a sink stream, it also needs to create a named pipe to write the respective media data. There are two different approaches to create and write multimedia data to the named pipe, depending on the considered media type: video or audio.

Unfortunately, OPAL does not offer a direct way to read the raw multimedia data that comes from the media stream and write it to a file (in this case, a named pipe). To circumvent this problem, it was necessary to direct the media stream to specialized sink media stream types. OPAL offers in its API a special class to sink raw PCM audio to a file. It also offers a way to export raw video data to an YUV file. Unfortunately, this solution only worked when writing to normal files (not named pipes). When it was the time to glue the Call Manager and the Flumotion together, instead of ordinary files it was required to use named pipes. However, with named pipes this solution did not worked. The problem was that the opening of the named pipes blocked. With that block in the open procedure, OPAL was set in a state corresponding to a deadlock. After many tries (i.e.
creating the named pipes with different permissions, running the application with different permissions, creating the named pipes within the OPAL Manager initialization, creating the named pipes after the creation of files (created by OPAL’s instances) and creating threads within processes to create the named pipes) it was finally discovered a bug in the OPAL Framework: when opening a file, OPAL does not consider the case of dealing with named pipes. Therefore it misses the NONBLOCK flag that is necessary when opening/writing named pipes.

The solution, again, was different for the audio and the video streams. Since OPAL offers the already mentioned AudioRawMediaStream class to export PCM raw audio data to a file, the solution was to descend this class and substitute its initialization and writing methods. In the initialization method, the file should be opened with NONBLOCK flag. In the write method of the descendent class (that was developed for the solution), it is written the raw data to file. The file, in this case, is a named pipe. It should also be noted that AudioRawMediaStream uses PFile to deal with the creation, writing and reading of files. As with OPAL Framework, PFile does not features the NONBLOCK flag. Therefore, the created descendent of AudioRawMediaStream (denoted MyRawMediaStream) bypasses the PFile and uses the standard unix system functions: open, write and read. This is why the write and read methods of AudioRawMediaStream were also redefined.

For the incoming video streams, the only feasible solution was to insert specific code directly in the OPAL Framework. The opening of the YUV file, in which OPAL writes the raw video data, is made in an internal class of PTLib Framework, which is also used by OPAL. The class is PVideoFile. However, the real writing is made in a descendant of this class: PYUVFile. In theory, the best solution would be to implement a direct descendant of the PYUVFile class and overwrite it with unix system functions to fit the named pipe requirement. However, since the PYUVFile is used in offer internal classes of the OPAL Framework, to implement that solution it would be necessary to implement descendants in all other classes, from the media stream classes to PYUVFile. Such modification would introduce many other problems and a further delay in the implementation of the final solution. Therefore, a more practical solution was opted instead:

- Inserting and/or substituting the open, write and read methods of both PVideoFile and PYUVFile. Such substitution uses UNIX system input/output functions with the NONBLOCK flag.

With the problem of named pipes solved, the last task related to the implementation of the Local Connection was to manage the launching of Flumotion instances. The first Flumotion instance is launched when the Local Connection has already created two media streams for both audio and video (correspondent to the speaker). The second Flumotion instance is created when another video media stream is created (correspondent to the presentation supporting media).
6.1 OPAL Call Manager Architecture

6.1.2 Call Flow

To provide a clearer vision on how the OPAL’s objects work together to implement the call manager, in this section it will be described the call flow procedure as it is illustrated in figure 6.2. Note that in this figure, it is only illustrated one media stream. In a real case scenario, it would be two media streams received (video and audio from the audience) and three media streams transmitted (two video streams and one audio stream from the speaker and the supporting media), in case of the audience side. The corded number in this figure represent interactions between the objects:

1. The Local Endpoint informs the Manager that it wants to establish a new call.

2. The Manager creates a Call instance that signals a remote client through the H.323/SIP Endpoint.

3. When the call is established the H.323/SIP Endpoint sends to the Call (through an H.323/SIP Connection) each media stream (H.323/SIP MediaStream) the will be used in the call.

4. For each media stream that is being sent by the remote client (in this figure it is only being illustrated one stream) the call instance asks the Local Connection to generate one Local MediaStream.

5. The Local Connection creates the Local MediaStream which will sink the media to a named pipe.
6. System Implementation

6. The Call instance will also create output MediaStreams to send to the remote client (H.323/SIP Endpoint). To do this is, it asks the Local Connection to create those MediaStreams.

7. The Local Connection creates a Local MediaStream. This Local MediaStream is responsible for acquiring media from a webcam, audio input, or any other input device.

8. The Local Connection sends the created MediaStream to the Call instance.

9. The Call instance redirects the MediaStream to the H.323/SIP Endpoint, that again sends the media streams to the remote client.

In figure 6.2 it is illustrated the local side of the Call instance in a much greater detail than the H.323/SIP side. Despite that, the process is similar to the local side. It was omitted from this figure in order to not overcomplicate it.

6.1.3 HTTP Server

The HTTP Server is a module inside the Call Manager as it was seen before, its job is to serve as a backbone to the interface module. The HTTP Server runs in a different port than the standard HTTP (80) and receives POST and GET requests from the Interface module.

The HTTP Server was implemented using PTLib HTTP classes. In particular, the PHTTPServer, class implements the HTTP Server which is initialized to listen to connections in a specified port. In this implementation, the server is initialized from port 8080 to above. This means that if port 8080 is busy, the server will try to listen on 8081, 8082, 8083, and so on.

The PHTTPServer uses virtual methods to process the received POST and GET requests. They are called by OnPost and OnGet. This is where all the server implementation takes place by implementing a specific handler for each of interface's website events: standby, ringing, settings, oncall, etc.

The OnGet method is used to pass dynamic information to the user. For example, when a remote user is calling, the interface displays its name. What really happens is that OnGet substitutes the "remote client" token that is present on the interface HTML file.

The OnPost method is used when there is a text form available for the user to submit. For example, when the user wants to place a call, he needs to insert the address of the remote client to be called. This address needs to be passed to the OPAL Manager. This is done using a form tag in the HTML code, that is then handled by the OnPost method.

6.1.4 Settings Storage

The Settings Storage module is nothing more than a way to store, in a persistent medium some video conference information such as what device to capture video/audio from. Since an
objective of this work is to provide a base for future development, the settings were implemented so that it would be easy to extend the way of how to persist them. This is illustrated in figure 6.3. To achieve this goal, an abstract class VCSettings was implemented with the methods: Load, Store, SetProperty and GetProperty. All the other classes that implement the storage of options will derive from this class and implement the Load and Store methods.

In this solution, a simple text file was used to extend the VCSettings class. It's name is VC-SettingsTXT. It simply stores each property in two text lines. The first line is the property name and the second line is its value. To handle the file writing and reading functions, it was also used a PTLib class: PTextFile.

With this design approach it is easy to change the way to store the data without changing the whole application. Furthermore, the settings are used in the OPAL Manager class. The manager class that was extended from OPAL to implement this solution has a VCSettings attribute. This attribute is then used to store changes to properties, as well as pulling them in the desired time (for example, when creating a media stream to capture audio).

6.2 Flumotion Stream Server Configuration

In this section it will be described how the configuration files for both the Flumotion Server and the Flumotion Worker were implemented.

6.2.1 Server Configuration

Figure 6.4 illustrates the processing pipeline that was implemented in the Flumotion Server. As it was discussed before, The Streaming Module is implemented by two Flumotion instances. This figure represents the particular Flumotion instance that is used with both audio and video
6. System Implementation

![Flumotion processing pipeline diagram](image)

Figure 6.4: Flumotion processing pipeline.

... streams. The other Flumotion instance pipeline is similar to this one, with the exception of the missing audio component, which will be used to transmit the supporting media video.

The first components of the pipeline are the producers. This particular Flumotion component enables a general GStreamer pipeline to be used instead of a Flumotion component. Since Flumotion does not have a default component to read from a file (named pipe) a GStreamer pipeline was used instead. There was a lot of time wasted in trying to read from a file since in the initial times of learning how to use Flumotion it seemed impossible to read from a file. The lack of information about the pipeline producer component delayed the implementation in several weeks.

The producer pipeline for the video is:

```
filesrc location=videofilename ! videoparse width=videowidth height=videoheight framerate=videoframerate ! ffmpegcolorspace
```

Videofilename, videowidth, videoheight and videoframerate are specific parameters of the stream. These parameters are defined by OPAL, when the Flumotion instance is launched.

The pipeline producer for the audio is:

```
filesrc location=audio.pcm ! audiosparse rate=audiorate channels=1 ! audioconvert
```

Since there is only one individual stream of audio, there is no need for a audiofilename. But the audiorate parameter is needed, to avoid OPAL to incorrectly define different rates, depending on what signaling protocol (H.323 or SIP) is used. They are either 16000 or 8000.

The pipeline producers for audio and video were the most complex parts of developed work, related to the server configuration file of the Flumotion Server. Those difficulties mainly arose by lack of documentation and led to some unexpected delays in the development of the presented work. All the other components are default components of Flumotion.

Besides the definition of the streams, Flumotion also requires a port to be used by the server, so that the Flumotion Workers can connect to it. This port is also defined by OPAL, upon the creation of the Flumotion Server configuration file.

6.2.2 Worker Configuration

The Flumotion Worker configuration was straightforward. Since it was not the objective of this thesis to implement a complex encoding system, only a Worker instance will be used for each server instance. All that it is needed to define in the worker configuration file is the port number in which the server instance will be listening to.
6.3 HTML Interface

In this section, it will be described the developed HTML Interface. As it was discussed in the previous chapters, the interface module is implemented by a web page. This way, users can use whichever browser it suits them more. Since the media streams are being delivered within a WebM container, which is natively supported by the HTML5 standard, every browser that adopts HTML5 will be able to play WebM multimedia streams that are being streamed from Flumotion. It is also an advantage to use this implementation approach because it is not dependent of the operating system (or Linux distribution).

The Web Interface is composed of two IFrames\(^1\). One for presenting context information and another one for presenting two multimedia streams that are being streamed through Flumotion. This separation was adopted so that when the user is on a call, the system can refresh the context information frame automatically without disturbing the multimedia streams on the other frame. The context information frame is composed of several web pages, namely:

- **Standby**: The standby page is the entry point to the system. It only has one single option to place a call through a text box. It also offers a link to the settings page that will be explained below. The standby refreshes itself from time to time (5 seconds), so that it can contact the HTTP Server embedded in the Call Manager module and update itself. This is useful when a call is being received. Upon a status change the standby page needs to dynamically and independently redirect itself to the Accept Call page. That is why the refresh is necessary and was the main reason for separating the context information from the multimedia video

---

\(^1\)If frame stands for inline frame and is used to embed another document within the current HTML document.
6. System Implementation

- **Settings**: The settings page enables the user to choose what devices should the system use to capture video/audio. The user should choose three interfaces. One for each of: speaker’s video, supporting media video and speaker’s audio. Speaker’s video is automatically used when a call is initiated. The supporting media video is only used if the user wants to send another video stream (e.g. presentation slides). Finally, the speaker audio device is the input device that should capture the speaker’s voice.

- **Ringing**: The ringing page is a transitory page that has the purpose of showing context information to the user. When this page is presented for the user, the user will know that the call that was placed by him is ringing on the remote side. This page refreshes itself every second, so that it can acquire from the HTTP Server the current status of the call.

- **Accept Call**: In the accept call page the user can accept or reject an incoming call. This page is automatically presented when the system is in standby mode. If the user chooses to accept the call, the system redirects to the "in call" page. If the user rejects the call, the system goes back to the standby mode and therefore redirects itself to the standby page.

- **In Call**: The in call page, illustrated in figure 6.5, is presented when the user has either accepted a call or a call has been accepted by a remote client. This means that the in call page only appears after the ringing or the accept call pages. The in call page offers two option to the user: add secondary video and end the call. Both of them are obvious. The add secondary video option enables the user to use the supporting media video device, defined in the settings page, to capture real time video and send it to the remote user. The end call option enables the user to terminate the call and redirect him to the standby page. As the standby and ringing page, the in call page also refreshes from time to time, to check whether the call has been terminated by the remote user or not.

### 6.4 Summary

In this chapter it was described how the proposed solution was implemented.

The call manager module, implemented by an OPAL application, is the main module and where most of the work was done. It implements the call management; the HTTP server; the settings storage; and the creation and writing process of the singular media streams that are received to named pipes.

It was also described the XML configuration files of the Flumotion instances to stream the speaker voice and video and the supporting media video.

Finally, it was described the HTML interface, what are the elements that compose the interface, how they interact with each other and how the interface interacts with the HTTP server.
implemented in the call manager module.
7

System Evaluation

Contents

7.1 Parameters Definition .................................................. 87
7.2 Tests Definition .......................................................... 88
7.3 Results ................................................................. 90
7.4 Solution’s states ........................................................ 91
7.5 Summary ................................................................. 92

86
To evaluate the implemented solution it is necessary to define the parameters that serve as an evaluation meter to how the final solution performs. Such parameters are essential to characterize the solution in terms of performance and correctness. After the parameters are defined, the solution must be tested against those parameters. The tests should be close to real case scenarios where the solutions would be used. The tests should also be made on a controlled environment to assure the results are completely dependent on the solution’s implementation, leaving off problems that can happen with the environment (e.g. network congestion or hardware malfunction).

In the next sections, both the parameters and the tests will be defined. After the tests are conducted, the final work is to discuss the results. Therefore the results are a cross between the parameters and the tests and serve as a proof to how the solutions work.

### 7.1 Parameters Definition

To test how the solutions works, there is a need to define the parameters that, having in account what type of use the developed application will have, can describe the solution in terms of performance and correctness measures. These parameters, again, should be chosen having in account the usage of the solution in its real usage cases. It is important to have the context of the solution in mind when defining such parameters.

#### 7.1.1 Measures

The parameters that are used to evaluate the solution can have different measure types. In this methodology most parameters will be binary. This means that the solution either works or not. Binary measures define the correctness of the solution.

It can also be necessary to define parameters which have a more than two possible results (as the binary measures). This parameters serve the purpose of defining what is the performance of the solution on it works correctly. In this methodology the only parameter that have a non binary measure, will have a temporal measure. This means that it will measure time to perform some action.

#### 7.1.2 Parameters

In this section it will be defined the parameters that characterize how the implemented application performs. Since the solution is a video conference solution to assist on presentations from a speaker, the parameters should evaluate all the features that are needed for such scenario. The parameters are:

- **Audio and Video Synchronization**: This parameter defines if there is synchronization problems between audio and video. In video conference this synchronization can also be
7. System Evaluation

called lip-sync. Lip-sync is the synchronization between what the speaker is saying and how the lips are moving. Ideally, it should be seen from the remote client that the lips correspond exactly to the words coming from the speaker’s mouth. The audio and video synchronization is a binary measure. Which means that either the user can derive the audio from the video or not. The synchronization on the other side does not need to be perfect. If there is a minimal delay between the video and/or audio which does not compromise lip-sync, the synchronization between audio and video is still considered valid.

- **Delay**: The delay measurement is a parameter to define the temporal delay from the captured audio/video to the reproduction of such media in the remote client. This parameter is a temporal measure that will be expressed in seconds. The delay parameter defines both the delay of reproducing the remote client’s media streams and the reproduction of the media streams captured locally that will be reproduced remotely.

- **H.323/SIP Compatibility**: To assure that the solution works properly there is a need to define a parameter that signals its compatibility with the remote client. This is a binary parameter, which means that the solutions is either compatible or not with the remote client its being tested with in terms of network protocols.

- **Codec Compatibility**: If the H.323/SIP compatibility is positive, another cause of failure is the incompatibility between the codecs of the solution and the codecs of its counterpart. This parameter defines if the solution is not compatible with another video conference solution due to codec incompatibilities. Again, this is a binary parameter that is either false or true. True in case of compatibility and false in case of incompatibility. It should be noted that this parameter can’t be evaluated if the H.323/SIP Compatibility parameter returns false.

7.2 Tests Definition

In this section it will be defined the tests that will be conducted in order to check the parameters that were defined above and to cross them both and derive the results. The tests will be divided in two major categories: self-contained tests and integration with other video conference solutions. The self-contained tests are tests that are performed with two instances of the solution implemented. The integration with other video conference solutions are tests that check the implemented solution compatibility with other video conference solutions.

It will also be described the environment setup in which the tests will be conducted, as well as the steps to perform the test.
7.2 Tests Definition

7.2.1 Self-Contained Tests

The self-contained tests are tests that are performed using two instances of the implemented solution. They are essential to check the correct behavior of the solution, since they evaluate simultaneously the receiving and placing of a call. They are also useful to make assumptions on how the solution performs in a standalone way (communicating with itself).

The environment setup used to perform these tests is limited to the hardware at disposal. However, two instances of the solution running in separate computers are needed. The use of two virtual machines in the same computer is not an option, due to the requirement of multiple web cameras and audio reproduction systems. Again, the closest to reality in a controlled environment the better. Thus, on one side it was used a MacBook Pro laptop running Ubuntu in a VMWare virtual machine. Connected to the VMWare was a Logitec Webcam that also features an audio input. The other used video input was the built in macbook camera. The other side of the setup consisted in a laptop running Ubuntu with a Logitec webcam (an identical to the one described above) and a video grabber for capturing the supporting media.

To control the network it was also used a router, which connected the two setups together using ethernet.

The tests that were performed using this environment setup were:

- To establish a connection between one instance to the other using just one video source.
- To establish a connection using two video sources.
- To make the same tests switching the instance sides.

7.2.2 Integration with Other Video Conference Solutions

The integration with other video conference solutions are the final test package which defines how the solution performs. Since it is the objective of the solution to interact with other different video conference solutions, these tests aims to test the integration with as many video conference solutions as possible.

The environment setup in this case is the same as the above. One MacBook Pro Laptop running VMWare with Ubuntu. The MacBook Pro is running the implemented solution. The other PC is running either Ubuntu or Windows, depending on the video conference solution to use. Again, a router will be used to connect both computer together using ethernet.

Since there is not a single software video conference solution that uses either SIP or H.323 and features dual-video functionalities (see chapter 4), it is not possible to test two video sources with

---

\footnotesize

1 A video grabber is device for capturing the screen of a computer from the VGA port and sending it over usb, as a webcam.
## 7. System Evaluation

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Implemented Solution</th>
<th>Ekiga</th>
<th>Lynphone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio and Video Synchronization</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Delay</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>H.323/SIP Compatibility</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Codec Compatibility</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

*Figure 7.1: Evaluation results table*

The testing process that was utilized to evaluate the implemented solution was:

- Call from the implemented solution to software video conference solution.
- Call from the video conference solution to the implemented solution.

Within each step, every parameters is evaluated. The bidirectionality is tested to assure that the integration is working on every possible case.

### 7.3 Results

The results are the final step in evaluating the implemented solution. When the tests are performed against the parameters the result is a table that crosses the tests and the parameters. The table representing this cross is illustrated on figure 7.1 and is called an evaluation table.

This evaluation table represents in lines the solutions that the implemented solutions was tested with and in columns the evaluated parameters. It should be noted that delay is marked in the table as a binary parameter but in fact has a temporal measure. In this table it is only illustrated if the delay was acceptable or not for a fluid communication.

The self-contained tests showed that the application can communicate successfully with itself. Both H.323 and SIP protocols worked seamlessly without any problems.

There was a transfer of video and audio from each side to the other so it can be stated that the set of codecs worked. This was a special test case, because the same solution is in both the receiver and the caller, the codecs set are the same and so is their version. Because of this, in normal conditions, the codec compatibility should match 100%.

The delay was the same from each side: 6 seconds. This is explained by a slow streaming speed of Flumotion and the time that the VP8 encoder takes to encode RAW video. Flumotion depends on large buffers to stream without breaks throughout the call. Since there is no way of setting the Flumotion buffers, nothing can be done about it. Another cause of delay is the VP8
7.4 Solution’s states

encoder. Since VP8 (and WeBM) is a recent open source standard, there isn’t a fast VP8 encoder available yet.

To finish the self contained test results, the synchronization between audio and video worked fine. There was no problem deriving the audio from the video or vice-versa. The synchronization was not perfect but it was very close, with few milliseconds of delay that didn’t disrupt lip-sync.

The results for Ekiga show that Ekiga is compatible in every way with the implemented solution. There is a full compatibility both in H.323 and SIP protocols. The codecs were also compatible both in audio and video. The synchronization (lip-sync) was as much as a degree in milliseconds. Which means that the user sees almost instantaneously the video and the audio. The delay between the implemented solution’s streams to Ekiga was 2 seconds. In the other part the implemented solution only received Ekiga’s streams at about 6 seconds after the streams were captured.

Finally, the results for Linphone were not so satisfactory. Firstly, Linphone only supports SIP protocol and from the tests the protocol worked fine when establishing new calls. The problem appeared when the media streams were created. There weren’t fully compatible video codecs so there were errors creating the media streams and therefore the call terminates abnormally. Since the call is not initiated correctly, the delay and synchronization between video and audio could not be tested. This would be solved if the correct versions of the codec packages were installed to match the Linphone codecs. Nevertheless, since Ekiga tests worked well, it is expected that these parameters would assume the same values as Ekiga tests.

7.4 Solution’s states

Throughout the testing process and the general use cases, the implemented solution can have multiple states that indicate in what call state the solution is (calling, in a call, standby, etc). In this section it will be presented, with screenshots, the different states that the user can see in a typical utilization case of the application, that were also seen while testing the implemented solution in the previous sections. These states are:

- **Standby**: In the standby state, the application is waiting for a call to be placed or waiting for a remote call.

- **Settings**: In the settings page, the user can define the devices that will capture the speaker/audience’ audio and video and the supporting media.

- **Calling**: In this state, the user has just placed a call and the application is waiting for it to be answered, by a remote user.
7. System Evaluation

![Speaker side of the implemented solution.](image)

- **New call**: When the application is waiting for a call to be answered (or rejected), it is in the new call state.

- **In a call**: In this state, the call is established and the speaker and its audience can collaborate.

- **Completed call**: In this state, a call has just been completed and the application is again, on a standby state.

- **Call failed**: If, for various reasons, the initiation of a call fails, the application goes back to the standby state with a message about the cause of failure.

The figures 7.2 and 7.3 illustrate what the speaker and the audience see in the interface, when a call has been established. The speaker sees its audience and the supporting media (or slides) so that the local audience can also see the supporting media. In the remote audience site, the interface displays the speaker and the supporting media. This way they have at their disposal all the information the local audience also have: the speaker's audio and video and the supporting media video.

7.5 Summary

This chapter focuses on evaluating the proposed and implemented solution. It defined evaluation parameters, such as: video and audio synchronization, delays and SIP/H.323 compatibility. It also described the tests to evaluate the defined parameters. The tests that were performed were: self containing tests (that tested the implemented solution with itself) and compatibility tests.
7.5 Summary

Figure 7.3: Audience side of the implemented solution.

(That tested the implemented solution with other videophone solutions). Unfortunately, it was not possible to use video conference applications, with dual video.

Finally, it was discussed the implemented solution performance by analyzing and evaluating the parameters when executing the tests.
Conclusion

Contents

8.1 Advantages ............................................. 95
8.2 Features ................................................ 96
8.3 Limitations .............................................. 96
8.4 Future Work ........................................... 97
8.1 Advantages

The proposed application is a video conference system particularly optimized for presentation purposes and implemented using only open source software. To develop the application, it was studied what are the real requirements of a presentation session and how a new video conference solution could integrate with already established video conference rooms.

The implemented system imposed several challenges that were hard to overcome. The main task that had to be done, was to integrate the different technologies into one single software package. Considering the different technologies that compose the final solution (such as OPAL, Flumotion and HTML) the challenge was to find the best way to connect them the best and efficient way possible. Thus, in the final application, it was decided to use named pipes (to connect the streams from OPAL and Flumotion), a custom web server (to take requests from the HTML interface) and WebM multimedia container (to present multimedia streams in the HTML interface without the need to install third party browser plugins).

As a result of the developed work, it was defined a test framework to evaluate the most important requirements of the system. Such requirements include the ability to communicate with itself and with other video conference systems. As an example, it was evaluated if both signaling protocols (H.323 and SIP) and considered codecs were compatible and if the call was correctly established. This tests also evaluated the call quality in terms of synchronization between the audio and video.

In this final chapter it will be presented the advantages of the proposed solution, taking into account the objectives previously defined in chapter 1. The features of the developed solution will be presented in terms of codecs and technical details. Over the course of the development procedure, there were several limitations that had to be considered in order to continue the implementation process. Finally, it will be presented a set of aspects that can still be improved in future developments of this project.

8.1 Advantages

In this section it will be summarized the set up advantages offered by the implemented system, when compared with other alternative in terms of the objectives defined in chapter 1 and in terms of the comparison table illustrated in figure 4.8.

The developed solution is entirely based in open source software. The use of open source frameworks, instead of custom developed and closed frameworks, guarantees that the the implemented application will always be operational even when major operating system updates occur. It also means that the solution does not have developing costs from frameworks or software tools.

The solutions adopts both H.323 and SIP protocols. This enables it to communicate virtually
8. Conclusion

with any other video conference application that also implement these protocols. As it will be men-
tioned in the limitations section, although this is not enough to have a fully working communication
system, it is the most important feature to communicate with other video conference systems.

The solution was developed by separating the interface from the application logic. This means
that the interface’s layout can be modified without altering the call manager module code.

Recalling figure 4.8 from chapter 4, which compares the current applications available nowa-
days in the public domain, it can be stated that the implemented solution checks every features
except the MCU feature (which is not needed for a presentation purpose). Hence, it is the only
free video conference that can perform dual video, which is a fundamental requisite for any video
conference system used for presentation purposes, as described in chapter 1. Since it is based
on open source software there are no costs involved in developing this video conference applica-
tion for the specific purpose of presentations. The dual video feature is also extremely important.
Without this feature it would be impossible to enable the simultaneous transmission of both the
speaker’s video and the supporting media.

8.2 Features

There are several features that characterize the implemented solution. First of all, the solution
is compliant with both H.323 and SIP call signaling protocols. Thus, since it is entirely based on
open source software, the set of available codecs are only dependent on the installation process.
It depends on what software (codec) packages are installed when deploying the application. Nev-
evertheless, although H.261 and Theora were the only ones that were considered in the evaluation,
the addition of new codecs, is only dependent on the correct installation of the correspondent
software packages. In what concerns the audio codecs, most of them worked seamless without
any problems. The audio codecs that were tested included: G.722, PCMU, PCMA, Speex and
GSM.

The solution features a plain and rather simple web interface, that can work with every web
browsers that are compliant with HTML5 (even though HTML5 standard is not finished yet). The
multimedia streams that are played in the web browser use the WebM multimedia container format
which, features a VP8 encoded video stream and a Vorbis encoded audio stream.

It also features dual video for both the transmitted and received video streams and one audio
input source. Finally, it is offered for the user to choose what input sources he wants for: the
speaker’s video, the speaker’s audio and the supporting media video.

8.3 Limitations

In this section it will be described a set of limitations that had to be made in order to overcome
complex problems that were considered out of the scope of this thesis.
8.4 Future Work

The choice of an exclusive use of Linux operating system, instead of a multi platform solution, was imposed by the use of Flumotion. Since Flumotion is only available for Linux systems, another multimedia streaming software for MS Windows environments needs to be found. However, since the first objective was to develop an application that could run on Linux operating systems (because they are open source and free), it was impossible (due to time constrains), to adapt the solution to a truly multi platform configuration.

The slight delay that is verified is also due to Flumotion. In fact, Flumotion imposes a delay in the reproduction of the live streams whenever the HTML5 video player does not start right away.

Another strong limitation that Flumotion and named pipes impose is the inability to start Flumotion as soon as a call is coming or being placed. Since Flumotion needs to know the properties of the incoming streams when it is being started, and the named pipes must have a reader (which is Flumotion) when being created (due to OPAL limitations), Flumotion needs to be started at the start of the system. This means that at this point the format of the streams must be defined without even knowing what kind of streams it will be received. This limitation means that this system cannot work with every other system in runtime, because Flumotion stream properties must be hardcoded in compilation time. The properties of the streams include: frame rate and size for video streams and channels and sample rate for audio streams.

Even though Flumotion imposes several limitations in the final solution, it was the only streaming server of the studied ones (GStreamer, and FFServer) that could stream WebM multimedia streams over HTTP.

8.4 Future Work

Although completely functional, the implemented solution is not yet in a finished state. To be fully usable in a real life application, some aspects still need to be improved, as it will be described below:

- **Start of the streaming server only after a call has started (and the media streams have already been created):** This will enable the system to communicate with every other video conference solution without being dependent on hardcoding the stream properties in compilation time, thus making the system compliant with distinct terminals. To implement this feature, the best alternative would be to abandon Flumotion and WebM and adopt another streaming server and a browser video plugin.

- **Prepare for multiple platform:** To open the system for multi-platform support it would be needed to find an equivalent software for the encoding and streaming of the WebM streams.

- **Include support for other protocols:** Even though new protocols like Google Talk and Skype are closed or private, they provide an API to interoperate with them. OPAL architec-
8. Conclusion

ture enables the addition of new protocols. Therefore, it could be possible to integrate Skype and Google talk in the solution.

- **Enable the communication to more than one audience rooms**: In the actual stage of the solution, the speaker can use two video channels (speaker and supporting media), but can only communicate with one remote audience room. Through the use of an MCU, it could be possible and very favorable to add more remote audience rooms, so that they can all interact with the speaker.

- **Stream the presentation for third party users**: In addiction to having video conference rooms that are actively participating in the presentation sessions (by having the possibility to interact with the speaker), it could also be possible to transmit the whole presentation session (video from the speaker, video from the supporting media and audio from the speaker and from the audience) to passive receivers.

- **Record a presentation session in a multimedia video format**: The recording of a session would enable other users that could not attend the presentation session to see it after it has finished. To implement this highly requested feature, the video streams would have to be merge into a single video stream. The audio would also have to be mixed into a single audio stream. When the two streams were generated the final step would be to encode them in a multimedia container, such as WebM.

- **Improve the call delay**: Since the delay is caused by Flumotion. The best alternative would be to use another streaming software and use a browser plugin to communicate with the given streaming software.

- **Improve the web interface**: There is a problem with the current interface. The problem is that it needs to refresh itself to update its state (through the communication with the HTTP Server). A possible solution for an improved web interface would be to use javascript and AJAX for the automatic actualization of the interface when the call manager receives new context information. The use of an Apache server, instead of a custom built one, and PHP would make the web interface even more robust.
Bibliography


