



INSTITUTO SUPERIOR TÉCNICO  
Universidade Técnica de Lisboa

# Web 2.0 nas Comunicações Multimédia SIP (IMS)

**Tânia de Almeida Serrano**

Dissertação para obtenção do grau de Mestre em:

**Engenharia de Redes de Comunicações**

## **Júri**

Presidente: Professor Luís Eduardo Teixeira Rodrigues  
Orientador 1: Professor Fernando Henrique Corte Real Mira da Silva  
Orientador 2: Engenheiro Paulo Chainho  
Vogal 1: Professor Paulo Rogério Barreiros D'Almeida Pereira

**Setembro de 2008**



## Acknowledgments

First, I would like to thank Professor Fernando Mira da Silva and Engenheiro Paulo Chainho for their help and advice along the development of the dissertation.

Second, I would like to thank to my parents and family for the values of life that taught me, and what makes me the person I am today.

Finally, and most importantly, I would like to thank to my boyfriend for all the support during the last years. He was who prevented me of quitting in the most difficult times, especially during this last year.

## Sumário

O principal objectivo desta dissertação é adicionar um conjunto de novas funcionalidades num serviço de conferência e colaboração na Internet. Este serviço está em desenvolvimento na PT Inovação e tem o nome de *Tagarela*. As novas funcionalidades permitem que, uma vez dentro de uma sala de conferência, os utilizadores possam partilhar os seus desktops (*desktop sharing*), ou um conjunto de imagens pré-adicionadas e seleccionadas (*slideshow sharing*), tendo sempre um canal de voz disponível.

No âmbito desta dissertação também foi desenvolvida uma análise de alguns webphones existentes. Esta análise foi realizada com o intuito de substituir uma solução comercial, que estava a ser utilizada, por uma solução *open source*. A análise dos webphones foi feita tendo em conta um conjunto de requisitos pré-definidos. Depois do processo de análise, a solução escolhida foi integrada na interface do serviço de conferência.

Ao longo desta dissertação serão apresentados, não só um conjunto de serviços de comunicação web, mas também como é que essas tecnologias foram integradas, por forma a enriquecer o *Tagarela* com novas funcionalidades. Por fim, através da realização de testes de usabilidade torna-se possível provar que as novas funcionalidades não trazem dificuldades acrescidas à utilização do *Tagarela*. Isto acontece porque, com um número reduzido de acções, é possível interagir com o serviço, por forma a usar as novas funcionalidades.

## Palavras-chave

Integração, Partilha de Informação, Conferência, Colaboração.

## Abstract

The main objective of this dissertation is to add a set of new features into a conference and collaboration service on the Internet. This service is in development at PT Inovação and is named *Tagarela*. The new features allows that, once inside a conference room, the users may share their desktops (desktop sharing), or a set of pre-added and selected images (slideshow sharing), having always a voice channel available.

In the scope of this dissertation it was also developed an analysis of some existing webphones. This analysis was made with the intention of replacing a commercial solution, which was being used, by an *open source* solution. The analysis of those webphones was done taking into account a set of pre-defined requirements. After this process of analysis, the selected solution was integrated into the conference service interface.

Along this dissertation are presented, not just a set of web communication services, but also how those technologies were integrated, in order to enrich the *Tagarela* with new features. At last, through the realization of usability tests becomes possible to prove that the new features do not bring any greater difficulty to the use of *Tagarela*. This happens because, with a reduced number of actions, it is possible to interact with the service, in order to use the new features.

## Keywords

Integration, Sharing of Information, Conference, Collaboration.

# Contents

Acknowledgments . . . . .	i
Sumário . . . . .	i
Abstract . . . . .	ii
Contents . . . . .	iii
List of Figures . . . . .	vii
List of Tables . . . . .	ix
List of Appendices . . . . .	xi
List of Acronyms . . . . .	xiii
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Goals . . . . .	1
1.2 Organization . . . . .	2
<b>2 State of the art</b>	<b>3</b>
2.1 Web 2.0 . . . . .	3
2.2 SIP . . . . .	3
2.2.1 Brief History . . . . .	3
2.2.2 Session Initiation Protocol (SIP) vs. HyperText Transfer Protocol (HTTP) . . . . .	4
2.2.3 Architectural Components . . . . .	5
2.2.4 Messages . . . . .	7
2.2.5 Transactions and Dialogs . . . . .	7
2.3 IMS . . . . .	8
2.3.1 Basic Architecture of IP Multimedia Subsystem (IMS) . . . . .	9
2.3.2 IMS Architecture of PT Inovação . . . . .	12
2.3.3 Centralized Conferencing (XCON) Standard . . . . .	14
2.4 Additional Concepts . . . . .	15
2.4.1 Applet . . . . .	15
2.4.2 Servlet . . . . .	16
2.4.3 Model-View-Controller (MVC) Pattern . . . . .	16
<b>3 Web Communication Services</b>	<b>19</b>
3.1 Tagarela . . . . .	19
3.1.1 Architecture . . . . .	19
3.1.2 Main Features . . . . .	22
3.1.3 Entities . . . . .	23

3.1.4	Web Interface . . . . .	24
3.2	WebHuddle . . . . .	25
3.2.1	Architecture . . . . .	25
3.2.2	Struts . . . . .	27
3.2.3	Entities . . . . .	30
3.2.4	Features . . . . .	30
3.2.5	Work With WebHuddle . . . . .	31
3.2.5.1	Standard Features . . . . .	32
3.2.5.2	Modified Features . . . . .	33
3.2.6	Current WebHuddle Application Programming Interface (API) . . . . .	34
3.2.7	WebHuddle and Network Address Translation (NAT) . . . . .	35
3.2.8	Capturing Image for Desktop Sharing . . . . .	35
<b>4</b>	<b>Implementation</b>	<b>37</b>
4.1	WebHuddle and Tagarela Integration . . . . .	37
4.1.1	Create New Conference . . . . .	38
4.1.2	Joining a Conference . . . . .	40
4.1.3	Inside of Conference Room . . . . .	42
4.1.3.1	Start Desktop Sharing . . . . .	42
4.1.3.2	Start Slideshow Sharing . . . . .	44
4.1.3.3	Join Desktop or Slideshow Sharing . . . . .	46
4.1.3.4	Slide Upload and Management for Slideshow Sharing . . . . .	47
4.1.3.5	Request Permission . . . . .	48
4.2	Webphone Integration . . . . .	49
4.2.1	Analysis of Webphone Software Development Kit (SDK)s . . . . .	50
4.2.1.1	Analysis of Abbeyphone . . . . .	50
4.2.1.2	Analysis of eyeP Foundation . . . . .	51
4.2.1.3	Analysis of Emansip . . . . .	52
4.2.2	Emansip . . . . .	52
4.2.3	Webphone Implementation . . . . .	54
<b>5</b>	<b>Tests</b>	<b>57</b>
5.1	List of Functional Tests . . . . .	57
5.2	Test Procedure . . . . .	58
5.3	Results and Analysis . . . . .	63
5.3.1	Number of Required Actions . . . . .	63
5.3.2	Duration of the Tests . . . . .	64

<b>6 Conclusions</b>	<b>67</b>
6.1 Future Work . . . . .	67
<b>Appendices</b>	<b>71</b>





## List of Figures

1	SIP communication stack. . . . .	5
2	Example of a SIP request message. . . . .	8
3	Example of a session establishment and termination using SIP. . . . .	9
4	Simplified view of the IMS architecture. . . . .	10
5	Basic architecture of IMS. . . . .	12
6	IMS architecture of PT Inovação ( <i>simplified</i> ). . . . .	13
7	Conferencing System Logical Decomposition. . . . .	14
8	Representation of the Model-View-Controller (MVC) structure. . . . .	17
9	Functional Architecture of <i>Tagarela</i> , before integration with <i>WebHuddle</i> . . . . .	20
10	Login page of <i>Tagarela</i> . . . . .	24
11	Architecture of <i>WebHuddle</i> . . . . .	26
12	Principal components of structure of the <i>WebHuddle</i> framework. . . . .	26
13	Example of code that allows to enable a controller servlet. . . . .	28
14	Example of an action defined in <i>struts-config.xml</i> . . . . .	28
15	Example of a form bean defined in <i>struts-config.xml</i> . . . . .	29
16	Definition of the location of the <i>struts-config.xml</i> . . . . .	29
17	Description of the buttons that the <i>moderator</i> can use during a slideshow sharing. . . . .	31
18	Interface of the <i>WebHuddle</i> Server, at the beginning. . . . .	32
19	Changes to the functional architecture of <i>Tagarela</i> , after the integration of the <i>WebHuddle</i> features (desktop sharing and slideshow sharing). . . . .	37
20	Interface to create new thematic conference. . . . .	38
21	Sequence diagram of the creation of a new conference room. . . . .	39
22	Page with the list of subscribed conferences. . . . .	40
23	Sequence diagram of joining a conference room, using a Webphone. . . . .	41
24	Sequence diagram of starting one desktop sharing, in <i>Tagarela</i> . . . . .	43
25	Sequence diagram of starting one slideshow sharing, in <i>Tagarela</i> . . . . .	45
26	Sequence diagram of joining one desktop/slideshow sharing, in <i>Tagarela</i> . . . . .	47
27	Sequence diagram of uploading images to use in the slideshow sharing, inside one <i>Tagarela</i> conference room. . . . .	48
28	<i>ModeratorConf</i> receives permission request. . . . .	48
29	Interface for <i>ModeratorConf</i> decide if wants to give permission or not. . . . .	49
30	Interface of the <i>Emansip</i> softphone. . . . .	54
31	Chart with the representation of the tests of usability result. . . . .	64
32	Chart representing the maximum, minimum and average duration of each test. . . . .	64

33	SIP message structure. . . . .	71
34	Request-line of the start-line. . . . .	72
35	Status-line of the start-line. . . . .	73
36	Interface to create new ad hoc conference. . . . .	75
37	<i>Tagarela</i> Floor Control. . . . .	76
38	Interface of a desktop sharing, with the description of the available buttons. . . . .	77
39	Example of a slideshow sharing page. . . . .	78
40	Example of the page that the user, who joined a desktop sharing, will see. . . . .	79
41	Example of the page that the user, who joined a slideshow sharing, will see. . . . .	80
42	Interface to upload and manage the slides to use in a slideshow sharing. . . . .	81

## List of Tables

1	Summary table with the result of the study of the webphones. . . . .	53
2	Summary table with the features of the libraries used by <i>Emansip</i> . . . . .	55
3	Mandatory headers. . . . .	72
4	Examples of request methods. . . . .	72
5	Types of response messages. . . . .	73
6	Classes of response messages. . . . .	73



## List of Appendices

A. More information about SIP	73
B. Example of tables from WebHuddle Server database	74
C. Interface to create new ad hoc conference	75
D. Description of the <i>Floor Control</i> area	76
E. Start Desktop Sharing	77
F. Start Slideshow Sharing	78
G. Additional Information to Join a Desktop/Slideshow Sharing	80
H. More about Upload Files to a Slideshow	81



## List of Acronyms

**3G** Third Generation

**3GPP** Third Generation Partnership Project

**3GPP2** Third Generation Partnership Project 2

**AAA** Authentication, Authorisation and Accounting

**ADSL** Asymmetric Digital Subscriber Line

**AJAX** Asynchronous JavaScript and XML

**API** Application Programming Interface

**AS** Application Server

**AV** Audio and Video

**B2BUA** Back-to-Back User Agent

**CCCF** Conference and Collaboration Control Function

**CDMA2000** Code-Division Multiple Access version of the IMT-2000 standard

**CMTS** Cable Modem Termination System

**COM** Component Object Model

**CRBT** Color Ring Back Tone

**CSCF** Call Session Control Function

**DLL** Dynamic Link Library

**DNS** Domain Name System

**DSL** Digital Subscriber Line

**DSLAM** Digital Subscriber Line Access Multiplexer

**DVB** Digital Video Broadcasting

**DWDM** Dense Wavelength Division Multiplexing

**EJB** Enterprise JavaBeans

**ETSI** European Telecommunications Standards Institute

**FMC** Fixed Mobile Convergence

**GPRS** General Packet Radio Service

**GSM** Global System for Mobile Communications

**GUI** Graphical User Interface

**HD** High-Definition

**HSQldb** Hypersonic SQL

**HSS** Home Subscriber Server

**HTML** HyperText Markup Language

**HTTP** HyperText Transfer Protocol

**ID** Identifier

**IE** Internet Explorer

**IETF** Internet Engineering Task Force

**iLBC** internet Low Bitrate Codec

**IMS** IP Multimedia Subsystem

**IP** Internet Protocol

**ISDN** Integrated Services Digital Network

**ISP** Internet Service Provider

**I-CSCF** Interrogating-CSCF

**J2EE** Java 2 Enterprise Edition

**JSLEE** Jain Service Logic Execution Environment

**JSP** JavaServer Page

**JVM** Java Virtual Machine

**LAN** Local Area Network

**LBS** Location Based Services

**MGCF** Media Gateway Control Function



**MGW** Media Gateway

**MMS** Multimedia Messaging Service

**MMUSIC** Multiparty Multimedia Session Control

**MPLS** Multi Protocol Label Switching

**MRF** Multimedia Resource Function

**MRFC** MultiMedia Resource Function Controller

**MRFP** MultiMedia Resource Function Processor

**MVC** Model-View-Controller

**NAT** Network Address Translation

**NGN** Next Generation Networks

**OLT** Optical Line Terminal

**OMA** Open Mobile Alliance

**OS** Operating System

**PCM** Personal Communication Manager

**PDA** Personal Digital Assistant

**PDF** Policy Decision Function

**PIN** Personal Identification Number

**PINT** PSTN and Internet Interworking

**PT** Portugal Telecom

**PTT** Push-to-talk

**PSTN** Public Switched Telephone Network

**P-CSCF** Proxy-CSCF

**QoS** Quality of Service

**RFC** Request for Comments

**RSM** Resources Share Management

**RTCP** Real-time Transport Control Protocol

**RTP** Real-time Transport Protocol

**SCTP** Stream Control Transmission Protocol

**SD** Standard-Definition

**SDK** Software Development Kit

**SDP** Session Description Protocol

**SDH** Synchronous Digital Hierarchy

**SGW** Signalling Gateway

**ShipNet** Service Handling on IP Networks

**SIMPLE** SIP for Instant Messaging and Presence Leveraging Extensions

**SIP** Session Initiation Protocol

**SIPPING** Session Initiation Protocol INvestiGation

**SLEE** Service Logic Execution Environment

**SMS** Short Message Service

**SMTP** Simple Mail Transfer Protocol

**SONET** Synchronous Optical Networking

**SPIRITS** Service in the PSTN/IN Requesting Internet Service

**SRTP** Secure Real-time Transport Protocol

**SS** Soft Switch

**S-CSCF** Serving-CSCF

**TCP** Transmission Control Protocol

**TISPAN** Telecoms & Internet converged Services & Protocols for Advanced Networks

**TLS** Transport Layer Security

**UA** User Agent

**UAC** User Agent Client

**UAS** User Agent Server

**UDP** User Datagram Protocol

**UE** User Equipment

**ULH** Ultra Long Haul

**UMTS** Universal Mobile Telecommunication System

**URI** Uniform Resource Identifier

**URL** Uniform Resource Locator

**VB** Visual Basic

**VCC** Voice Call Continuity

**VoIP** Voice over IP

**VOW** Voice Over Web

**VPN** Virtual Private Network

**WAR** Web application ARchive

**WLAN** Wireless Local Area Network

**XCON** Centralized Conferencing

**XML** eXtensible Markup Language

**XMPP** eXtensible Messaging and Presence Protocol



# 1 Introduction

The recent evolution of Internet provides a set of features that allows people to share all kinds of information, using several types of applications. Nowadays, there are applications that allows the users to share their pictures (E.g., *Hi5* [1]), their movies (E.g., *YouTube* [2]), and even their own thoughts (E.g., *Blogger* [3]). It is also possible for users to choose how they want to organize their webpages (E.g., *iGoogle* [4]), adding and removing items, changing the webpage themes, etc. Now more than ever, is important to develop new applications that allow users to create, manage and share the information that exists on the Internet.

We live in an age where is increasingly difficult to interact with people personally, because the people are mostly occupied. So, it is important to have a service that allows people to communicate, not just by writing text on a page, but also speaking and seeing the person with whom they are talking. Together with all these factors, the communication also has to be at the lowest possible cost. Now, imagine these features used not just to talk with one person, but to talk with a lot of people, at the same time, wherever they are. This whole set of features turns *web communications* more personal and attractive.

Taking all this into consideration, this dissertation aspires to investigate and try the using of Web 2.0 technologies, in the development of Service Delivery Platform (SDP), for the ALL-IP networks (e.g., Third Generation Partnership Project (3GPP) IMS). For this work, it is also one aim the investigation and testing of Web 2.0 concepts (e.g., creation/generation of services and contents by the users themselves) on the services platforms to the ALL-IP 3GPP IMS networks. This will lead to the development of new components to facilitate the creation of collaborative and social applications, applied to multimedia communications.

## 1.1 Motivation and Goals

Instead of having an open application for audio and video conversation, another application for chatting, and yet another one for file sharing, with this new service the user only needs to open a browser. This does not only saves computer resources, as yet provides a more organized work environment. This type of collaboration and conference services allows that, with only an Internet browser, the user can interact with several people.

So, the main goal of this dissertation is to add features to a service, that has already available some resources, such as audio conversation, file sharing, video sharing and chat service. The new features will enrich the service providing two more applications, from where the users can share data with others participants, in a collaborative environment.

With the intent of reduce even more the number of needed applications, to interact with the service, another goal is to integrate a webphone. This will avoid the requirement of having a softphone opened to enter in a conference room and to communicate with the others users.

This dissertation can also be an important contribution for companies that have offices throughout the country, for example. Imagine that there is an employee of the company that is at an office at 300 km from the headquarters of the company, and must have weekly meetings with the Director. Instead having to go to meetings, the employee may use the service to start a slideshow sharing with the Director, and make the presentation of the weekly reports. This will save time travel of the employee, as well as the money of the company spent on these trips.

Another example of utilization of the new service is the case of a Call Center, from an Internet operator. When a customer calls to the Call Center, with a question about how to install a program on the computer, the Call Center employee may use the desktop sharing. Using this resource, it is possible to show to the customer what is needed to do, in order to install the program.

## 1.2 Organization

The remainder of this dissertation is organized into five main chapters. The following chapter, chapter two, provides an introduction to the Session Initiation Protocol (SIP) protocol. This chapter also provides an overview of the state of the art of IMS architecture, in particular the conceptual IMS architecture of PT Inovação. At the end of chapter two are explained some additional concepts, that will be used in the following chapters.

Chapter three provides information about two web communication services: *WebHuddle* and *Tagarela*. For each of these services is presented their architecture, main features and the entities that have. In chapter four is presented how the implementation of the new features in *Tagarela* were made. This chapter also provides information about how one webphone was added to *Tagarela* interface.

Chapter five consists of the presentation of a list of functional tests, that were made to prove that the service works. Finally, chapter six draws some final conclusions and lays out foundation for future work in this area.

## 2 State of the art

In this section, the actual state-of-the-art of SIP protocol and IMS architecture are reviewed. In the first place will be presented a brief explanation of what is the Web 2.0. After, a summary will be presented about how did SIP first appears, and its workflow will be also presented. Subsequent, an IMS architecture will be presented, as well as some of its concepts.

### 2.1 Web 2.0

The concept of "Web 2.0" was born of the idea that the web is increasingly important, because of the exciting new applications and sites that are emerging at the present-day [5]. Today, the term "Web 2.0" has been widely adopted to describe a new web stage where web applications are increasingly more collaborative and depend on contents distributed by the final users, instead of conventional static information repositories.

However there is still a disagreement about what Web 2.0 really means, because some people believe that the term is just a marketing buzzword and others believe that Web 2.0 is the new conventional wisdom.

Some examples of applications in Web 1.0 and Web 2.0 are Britannica Online (Web 1.0) vs. Wikipedia (Web 2.0), Personal websites (Web 1.0) vs. Blogging (Web 2.0), and Content Management Systems (Web 1.0) vs. Wikis (Web 2.0).

### 2.2 SIP

#### 2.2.1 Brief History

SIP was originally developed by the Internet Engineering Task Force (IETF) Working Group, called Multiparty Multimedia Session Control (MMUSIC) [6]. The first version of this protocol (Version 1.0) was submitted in 1997, as an Internet Draft. In 1998 some significant changes were made, resulting on the second version of the protocol (Version 2.0). In March 1999, the protocol achieved the status of Proposed Standard and it was published, in April, as Request for Comments (RFC) 2543 [7].

In September 1999 the IETF established one working group about SIP. This happened because of the growing interest on SIP protocol. In July 2000 it was delivered one Internet-Draft containing bug fixes and some clarification to SIP. This document was named RFC 2543 "bis". Later, it was published as RFC 3261 [8], replacing the original specifications of RFC 2543. However until now several RFC have been made as SIP extensions.

The growing popularity of SIP in the IETF allowed the establishing of others working groups related to SIP. One of these cases is the Session Initiation Protocol INvestiGation (SIPPING), that has the purpose of investigate new applications of SIP and develop requirements for new SIP extensions. Another group

that was established was SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE). This group is responsible to standardize related protocols for presence and instant messaging. There are other groups that use SIP protocol like, for example, Public Switched Telephone Network (PSTN), PSTN and Internet Interworking (PINT) and Service in the PSTN/IN Requesting Internet Service (SPIRITS).

SIP includes some elements of two well-known Internet protocols: HyperText Transfer Protocol (HTTP) [9], that is used for web browsing, and Simple Mail Transfer Protocol (SMTP) [10] for e-mail. From HTTP SIP uses the client-server model, as well as the usage of Uniform Resource Locator (URL) [11] and Uniform Resource Identifier (URI) [12]. Of SMTP, SIP applies the text encoding scheme and the headers style that are used on SMTP messages. Some examples of these headers are *To*, *From* and *Subject*.

An advantage for SIP from using these elements is related with the possibility of extensions. That is, SIP can be changed to support new features and services, as call control services, mobility and interoperability with the telephony systems that already exists.

### 2.2.2 SIP vs. HTTP

While SIP is based on HTTP, there are some differences between these two protocols [13]. While HTTP allows the content integration (text, audio, video, etc) on web pages, SIP performs the integration of different media contents on the session. SIP use the request/response paradigm that is used by HTTP, as well as headers and codes. One example of used header is "404: *Address not found*". This header represents one error when one page is not found.

However, there are some key differences between SIP and HTTP. Unlike HTTP, SIP is a peer-to-peer protocol and a web server, using HTTP, does not originate requests. Besides, SIP can generate multiple responses, and for multiple destinations, through one single request. Another difference, on the architecture point of view, is that the HTTP services are, typically, hosted in HTTP servers. These servers generate responses to the received requests. For the other hand, SIP servers (proxy server, Redirect Server and Registrar) make the service routing.

As was already told, SIP uses URIs that allow us to identify the participants and the resources existing in one communication session. The SIP URI is similar to an URL *mailto*, and its usual form is:

sip:user:password@host:port;uri-parameters?headers

Besides this format, there are another SIP URIs like:

- Phone call directly to a PSTN phone:
  - sip:1-123-456-7890@redepstn.pt;user=phone;

The SIP parameters can be used to provide additional informations. In this example, *user=phone* indicates that this call is made to a phone number.



- Internet call:

- sip:123-456789@ist.utl.pt;user=phone; context=privnet;

The parameter *context=privnet* indicates that we are using a private network.

- Phone call directly to a computer:

- sip:alice@ist.utl.pt

To perform the translation of SIP URIs to Internet Protocol (IP) address, ports and transport protocols, Domain Name System (DNS) is used. DNS is also used to allow servers sending responses to backup clients, when primary clients fail. For peer-to-peer functionalities and services, like Voice over IP (VoIP), SIP interact with others protocols. Session Description Protocol (SDP) [14] defines the format to perform the characteristics and parameters description of the multimedia sessions.

Another used protocol is Real-time Transport Protocol (RTP) [15] that defines the standard format of the packets, to sending audio and video, through Internet. For the other hand, RTP provides out-of-band control information about the quality of the RTP data flow. At figure 1 is a representation of the SIP communication stack.

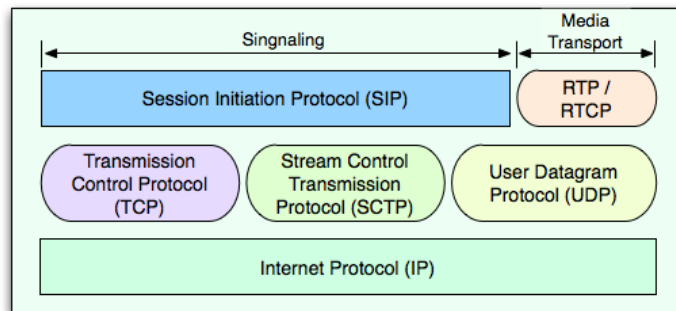


Figure 1: SIP communication stack.

### 2.2.3 Architectural Components

In this section it will be described the various components that make part of SIP [16]. A SIP network have five entities and each of these entities have specific functions, and they can be part of a SIP communication. The entities can act as a client (making requests), as a server (answering requests), or act as both. One physical device can have the functionalities of more than one SIP entity. This means that a network server that is a Proxy can also assume the Registrar functionalities.

Generally, it can be said that SIP networks have two basic components: SIP User Agent (UA), that start and answer calls, and SIP Network Server. The SIP architecture defines the next functional elements:

- **User Agent:** A SIP UA is a device that can start or receive SIP calls. These devices can be a terminal like a mobile phone, a Personal Digital Assistant (PDA) or a laptop, or even an endpoint like a answering machine. SIP supports peer-to-peer and client-server architectures. UAs act like peers, because they negotiate the session characteristics.

To start and terminate sessions, UA exchange requests and responses between them. In [8] UA are defined as being an application that contains an UA server and an UA client:

- User Agent Server (UAS): it's a server application that contact the user, when receives a SIP request. When this happens, it sends a response in name of the user;
  - User Agent Client (UAC): it is a client application that sends a SIP request.
- **Proxy Server:** The SIP proxy server is a fundamental component of the SIP infrastructure, and it has routing capabilities and support functions like authentication, accounting, register and security. The SIP proxy server is the first entity that receive all the SIP UA requests. Then, redirects this request to the SIP UA that have been called. Normally, there are two SIP proxy server - one on the caller side and other on the callee side. If necessary, the proxy rewrites the request messages before forwarding them to the servers.

Proxy Servers can be configured to use stateless and stateful transactions. Proxy Servers that are stateless receive requests and just sends the responses without saving any information about the transaction that have just been performed. On the other hand, stateful Proxy Server saves the information of all received requests, as well as the servers responses, and all messages that have the servers as origin. Finally, it should be mention that the SIP infrastructure can use a combination of proxy servers of the two kinds (stateless and statefull);

SIP Proxy Servers have one important function. This is, they can send several equal messages at once, to multiple devices. In the case of a user that is registered in more than one location, the forking Proxy Server will send one SIP INVITE message for each of these locations. When a server receives a SIP OK message, from one of the locations, it sends one SIP CANCEL message to the remaining locations. But to this function be available it's needed that the proxy server is configured to perform stateful transactions;

- **Redirect Server:** Redirect Server, as the name says, it's a server that accepts SIP request and answers with a address, for which the SIP message should be forward. Besides, it maps a destiny address, in SIP message, to one or more addresses, and gives a new list of addresses. This location information is on a database in the SIP Registrar. Unlike Proxy Server, Redirect Servers can't forward the request to others servers;
- **Registrar Server:** This functional element of the SIP architecture receives the users register requests, through Register messages. In other words, Registrar is a information location repository of

UAs. This element receives the UA register requests and replace the information on the location database. This information consists of the SIP address and the IP address associated;

The SIP REGISTER message will inform the Registrar, and consequently the network, about which address, or multiple addresses, where the user will be available. When another location change is made, the UA must send another SIP Register message to the Registrar. The location information that is saved at the repository is then used by the SIP proxy servers, to obtain the UA location of the callee;

- **Back-to-Back User Agent (B2BUA):** B2BUA is an entity that receives the requests, process them as if they were an UAS and, to determine how the request must be replied, act like an UAC and generate requests. This entity must maintain a state of calls and participate on the requests and responses transmission. B2BUA have a highest control of a call than the proxy. This happens because Proxy can't disconnect a call or change the messages.

#### 2.2.4 Messages

As it was already told before, SIP signaling is used to start, modify and terminate communication and collaboration sessions. This is all made through messages exchange. There are two kinds of SIP messages:

- **Request:** the client send a message to the server, to initiate an action (see figure 2);
- **Response:** message that is a reply to a request. It also indicates the status of the process.

More information about SIP can be found in Appendix A.

#### 2.2.5 Transactions and Dialogs

A SIP transaction is a group of messages that are exchanged by SIP components. A transaction starts when one of the components sends one request message. When a component receives a request, the correspondent response messages is sent. To process SIP transactions it's necessary to keep state information. Because of this, SIP components need to be *stateful*. This means that the components obtains the identifiers of the transactions, which are part of the SIP messages, and the components will update the state information, of all transactions that occur, with them.

SIP dialogs are relationships between two SIP endpoints (*peer-to-peer* relationships). This allow us to have a context for the sequencing and routing of the messages that are exchanged between SIP UAs. All dialogs are identified by the next fields at the *header* of SIP messages: *Call-ID*, *From* and *To*. All messages of the same dialog have always the same value on these three fields. The field *CSeq* maintains the sequence of all messages that are exchanged within a dialog. This field is increased automatically when a request is made. With this the transactions are identified within a dialog. On figure 3 is an example of SIP transactions and dialog, and on figure 2 there is an example of an INVITE message.

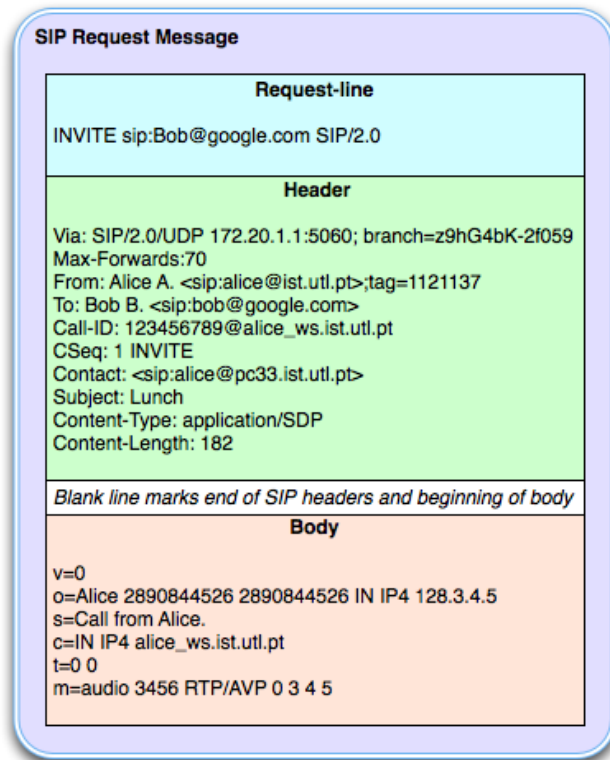


Figure 2: Example of a SIP request message.

## 2.3 IMS

Currently, an important architecture that uses SIP is IMS. IMS is an overlay service network architecture that can be applicable to any kind of IP network as, for example, Code-Division Multiple Access version of the IMT-2000 standard (CDMA2000), Global System for Mobile Communications (GSM)/General Packet Radio Service (GPRS), Universal Mobile Telecommunication System (UMTS) and Wireless Local Area Network (WLAN) [17]. It is a global standard that is supported by 3GPP, Third Generation Partnership Project 2 (3GPP2), European Telecommunications Standards Institute (ETSI), Open Mobile Alliance (OMA) and IETF, and it is often considered as the Universal Service Delivery Platform for Next Generation Networks (NGN), also supporting Fixed Mobile Convergence (FMC). Because there is no real IMS services that are standardised, for now IMS should only be considered as a service enabler.

A more specific definition of IMS is that it is a platform for multimedia service control. It combines real time resources, such voice and video-conference, with non-real time services, independently of the radio technology that is used. The IMS architecture intends to provide the next group of advantages:

- support to sophisticated multimedia services;
- session oriented connections;

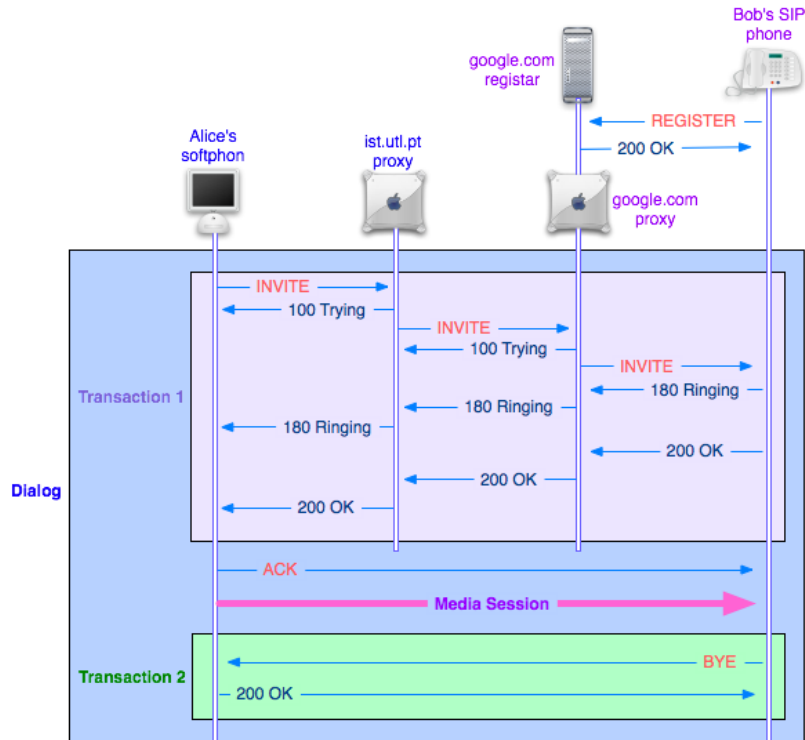


Figure 3: Example of a session establishment and termination using SIP.

- mobility with no restrictions, allowing also Home Control;
- Quality of Service (QoS);
- FMC of services and network operations;
- support of Legacy services;

One of the greatest benefits of the IMS architecture is the possibility of introducing sophisticated services, for subscribers. The actual networks already allow the provision of some of these services, but the following limitations occurs:

- low interaction between service platforms;
- low efficiency in databases administration, since each service platform often requires it own subscribers database, for provisioning. Obviously this is not the best way to implement and work with new services.

### 2.3.1 Basic Architecture of IMS

The IMS architecture is divided basically in tree layers (see figure 4):

- **Application Layer:** contains the services platforms (e.g. Push-to-talk (PTT), Location Based Services (LBS), Short Message Service (SMS)/Multimedia Messaging Service (MMS), etc.);
- **Control Layer:** responsible for control, including sessions establishment. The Soft Switch (SS) is the principal element of this layer;
- **Access Layer:** access media, including wireless interfaces (e.g. CDMA2000 and UMTS) and cable interfaces (e.g. Asymmetric Digital Subscriber Line (ADSL)).

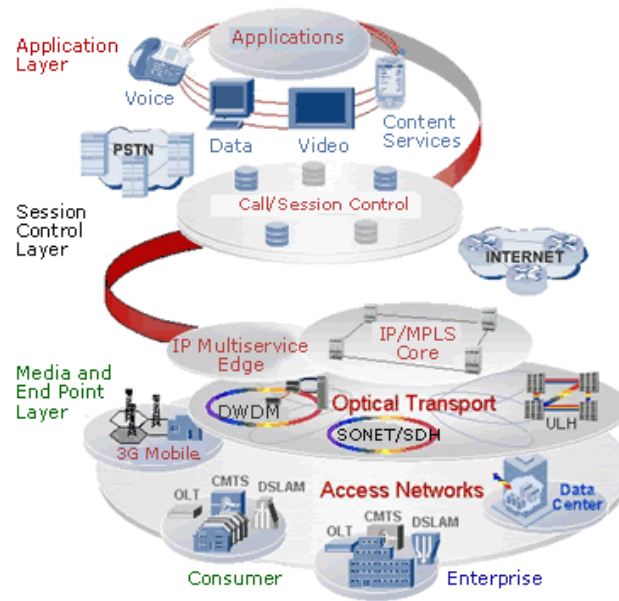


Figure 4: Simplified view of the IMS architecture.

SS have a main role in the IMS architecture. The SS contains the IMS server functions, being responsible for the call/session control, provided by the IMS, on the subscriber network (Home Network). SS manager IP sessions, provide the services, coordinates the session control with the others network elements, and allocates media resources.

On figure 5 it is shown a representation of the components of the IMS architecture [18]. The three majors components of this architecture are the following:

- **Proxy-CSCF (P-CSCF):** it represents the first contact within the IMS. The P-CSCF acts like a Proxy (as defined in [8] or following versions). This means that the P-CSCF either accepts requests and services internally or it forwards them. This IMS component must not change the request URI in the received SIP INVITE message. P-CSCF can also act like an UA (as defined in [8] or following versions), because it may terminate SIP transactions, as well as generate them. The P-CSCF has the following functions:

- forward the SIP register request received from the User Equipment (UE) to an entry point, determined using the home domain name, as provided by the UE;
  - forward SIP messages received from the UE to the SIP server (e.g. Serving-CSCF (S-CSCF)), whose name the P-CSCF has received as a result of the registration procedure;
  - execute the control policy set by the network operator;
  - coordinates with the access network, allowing the resources control and quality of the calls/sessions (QoS);
  - the operators can locally offer services controlled by the P-CSCF;
  - should realize SIP message compression/decompression.
- **Interrogating-CSCF (I-CSCF)**: this is the contact point in an operator’s network for all connections that are to an user of that network operator, or a roaming user that is actually located in that network operator’s service area. It may exist multiples I-CSCF in one network. The executed functions by the I-CSCF are:
    - designating of a S-CSCF to an user, executing a SIP register;
    - routing a SIP request received by other network, to a S-CSCF;
    - obtaining from the Home Subscriber Server (HSS) the S-CSCF address;
  - **S-CSCF**: is the manager of the SIP session and coordinates, with other network elements, the call/session control. The S-CSCF is responsible for executing the following functions:
    - SIP register: processes the SIP register requests;
    - session control: executes the call/session establishment, modification and termination;
    - service control: interacts with the Application Server (AS) to services and applications support.

But these are not the only components of the IMS architecture that exists. The others components, that are represented on figure 5, consist of the following:

- **HSS**: Contain the primary database with the data of all users, including authorized services, to which the several logical control entities (Call Session Control Function (CSCF)) may access to manage the subscribers. The user’s data will be forwarded to the S-CSCF. It also stores the temporary information with the S-CSCF location where the user is registered, at a given time;
- **AS**: provides service control for IMS and it may be directly connected to the S-CSCF. The AS interacts with the HSS to obtain subscriber profile information. It can also suport applications such as presence and conference control;

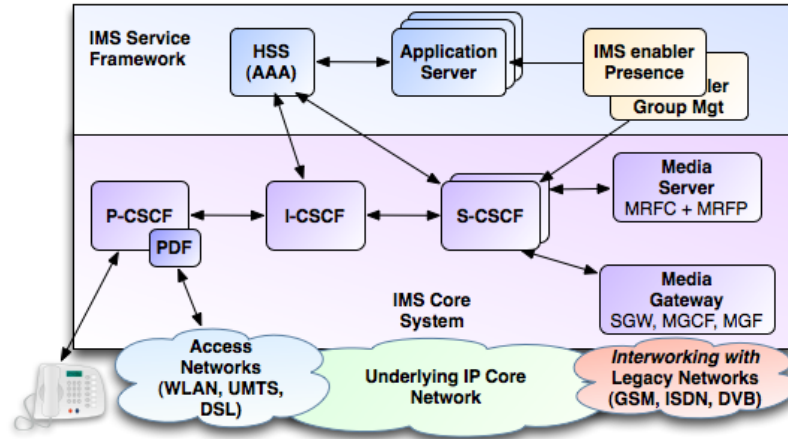


Figure 5: Basic architecture of IMS.

- **MultiMedia Resource Function Controller (MRFC)**: controls the media resources of the MultiMedia Resource Function Processor (MRFP). An example of these resources are the ones required to support a multi-user conference. Another functionality of the MRFC is to interpret the information coming from an AS through the S-CSCF and to control the MRFP accordingly;
- **MRFP**: is controlled by the MRFC and have the following functions:
  - available resources to be controlled by the MRFC;
  - join incoming media streams, from various parties;
  - processes media streams like, for example, media analysis;
  - notify the MRFC when an event has occurred.
- **Signalling Gateway (SGW)**: provides the signalling conversion in both directions, on transport layer, between SS7 and based-IP signalling;
- **Media Gateway Control Function (MGCF)**: has the function interworking of signalling between the elements of the IMS network and the Legacy networks (PSTN). The MGCF controls a group of Media Gateway (MGW) through H.248 signalling [19];
- **Policy Decision Function (PDF)**: it is the logical function that implements the decision about the policy to be applied. To do this, it uses QoS mechanisms at the IP connectivity layer.

### 2.3.2 IMS Architecture of PT Inovação

Since this Thesis was developed in collaboration with PT Inovação, PT IMS architecture will be presented in this section (see figure 6) [20]. This architecture includes the following components:



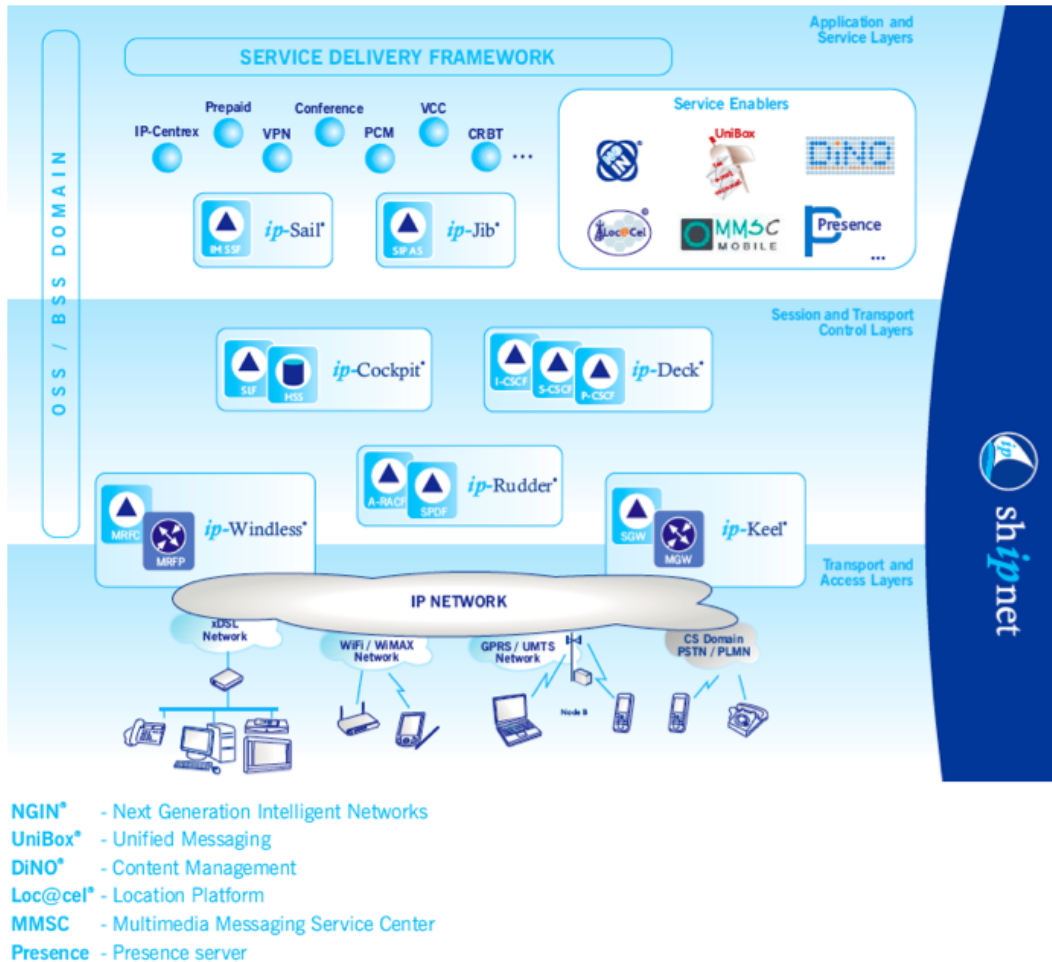


Figure 6: IMS architecture of PT Inovação (*simplified*).

- *ip-Sail*: solution that provides the operator with a flexibility tool for the deployment of a NGN service, with a high level of confidence in the charging method;
- *ip-Jib*: is a Next Generation Service Delivery Platform [21] and SIP AS, based on Jain Service Logic Execution Environment (JSLEE) (Service Logic Execution Environment (SLEE)) [22] and Java 2 Enterprise Edition (J2EE) technologies;
- *ip-Cockpit*: based on a modular and distributed architecture in order to achieve high performance, scalability and fault-tolerance levels. With HSS functions its modular approach allows a phased implementation, allowing customers to grow up the solution depending on its needs;
- *ip-Deck*: is the core session control infrastructure in PT Inovação's IMS compliant end-to-end service handling architecture Service Handling on IP Networks (ShipNet). It implements the three core IMS Call Session Control functions: P-CSCF, S-CSCF and I-CSCF;

- *ip-Windless*: is a flexible, open architecture multi-service platform that offers Telecom Operators and Service Providers a wide range of possibilities for delivering enhanced services over PSTN, converged and *all-IP* networks. The *ip-Windless* Platform allows for the fast development of new interactive voice and video applications, with efficient resource sharing;
- *ip-Rudder*: responsible for the admission control of new network flows and for managing its resources, ensuring the QoS;
- *ip-Keel*: allows the interworking with legacy networks.

### 2.3.3 Centralized Conferencing (XCON) Standard

An important standard in the IMS Conference Services is the standard Centralized Conferencing (XCON) [23]. A centralized conference is an association of endpoints (conference participants), with a central endpoint (conference focus). The focus has a direct relationship with the participants, through the maintenance of a separate call signaling interface with each of them. So, in the centralized conferencing model, the call signalling graph is always a star.

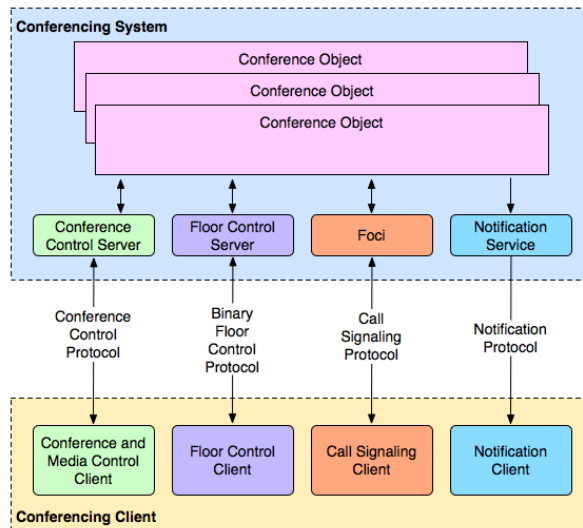


Figure 7: Conferencing System Logical Decomposition.

The centralized conferencing system proposed by [23] is built around an important concept of a conference object. This object provides the data representation of a conference during each stage of a conference (for example, creation and reservation). A conference object is accessed via the logical functional elements, with whom a conferencing client interacts, using the several protocols represented in figure 7. As it can be seen in this figure, the functional elements defined for a conferencing system described by the framework are: Conference Control Server, Floor Control Server, any number of Foci and Notification Service. Concerning to the protocols used, they are:

- **Conference Control Protocol:** provides the interface between a conference and media control client and the conference control server;
- **Floor Control Protocol:** provides the interface between a floor control client and the floor control server. E.g., Binary Floor Control Protocol;
- **Call Signaling Protocol:** provides the interface between a call signaling client and a focus. E.g., SIP, H.323, etc.;
- **Notification Protocol:** provides the interface between the conferencing client and the notification service. E.g., SIP Notify [24].

A conferencing system can support a subset of the conferencing functions represented in figure 7. But there are some components that may be used by other advanced functions. An example is the Notification Service that is used to relate information, such as the list of participants with their media streams, among several others components.

Concerning with the media graph of a conference it can be centralized, decentralized or any combination of both. When it is used a centralized media graph, the media sessions are established between a media mixer controlled by the focus and each of the participants. If it is used a decentralized architecture, the media graph is multicast or multi-unicast mesh among the participants. In this case, the media processing can be controlled by the focus or by the participants.

## 2.4 Additional Concepts

Before introducing and explaining the web communication services, and how those services were integrated, some additional concepts must be explained to better understand the implementation phase.

### 2.4.1 Applet

The applet is a program written in Java that can be included in a HyperText Markup Language (HTML) page. This is done, basically, in the same way that an image is included in a page. When using a browser that allows displaying Java technology, in a page containing an applet, the applet code is transferred to the client system and is executed by the browser Java Virtual Machine (JVM). An applet is delimited by the *applet* tag.

Applets are written in a language that is different from scripting or from HTML language that invokes it. An applet is written in a compiled language, usually java, while the container scripting language is a interpreted language, thence the higher performance or functionality of the applets.

The applet Application Programming Interface (API) allows to take advantage of the relation that exists between applets and web browsers. Applets can use the API to:

- be notified by milestones (*init*, *start*, etc);
- load the specific files concerning to the applet URL or the page where is running;
- show small status frames and make a browser to show a document;
- get parameters specified by the user on tag *applet*.

### 2.4.2 Servlet

The servlet is an object that runs in the HTTP server side, and that receives requests and generates responses, based on those same requests. The basic package of servlet define Java objects, to represent servlet requests and responses, like objects to reflect the servlet configuration parameters and execution environment.

The package *java.x.servlet.http* defines specific HTTP subclasses of generic servlet elements, including session management objects, that track multiples requests and responses between web server and client. Servlets can be encapsulated in Web application ARchive (WAR) files or as web applications, and can be generated automatically by JavaServer Page (JSP).

The Java Servlet technology provides the web developers with simple and consistent mechanisms to extend a web server functionality, and access to existing business systems. A servlet can be seen as an applet running on the server side.

The Java Servlet API allows software developers to add dynamic content to a web server, using the Java platform. The generated content is usually HTML, but it can also be eXtensible Markup Language (XML) data. The servlets can keep the state between server transactions, using HTTP cookies, session variables and URL rewrite.

### 2.4.3 Model-View-Controller (MVC) Pattern

An important architectural pattern used in software engineering is MVC [25]. By applying this pattern it is possible to separate core business model functionality from the presentation and control logic that uses this functionality. Such separation allows multiple views to share the same enterprise data model, which makes supporting multiple clients easier to implement, test and mantain.

As it can be seen in figure 8, the *Model* represents the information (the data) of the application and the business rules used to manipulate the data. The *View* corresponds to elements of the user interface such as text, checkbox items, etc. Finally, the *Controller* manages details involving the communication to the model of the user actions, such as keystrokes and mouse movements.

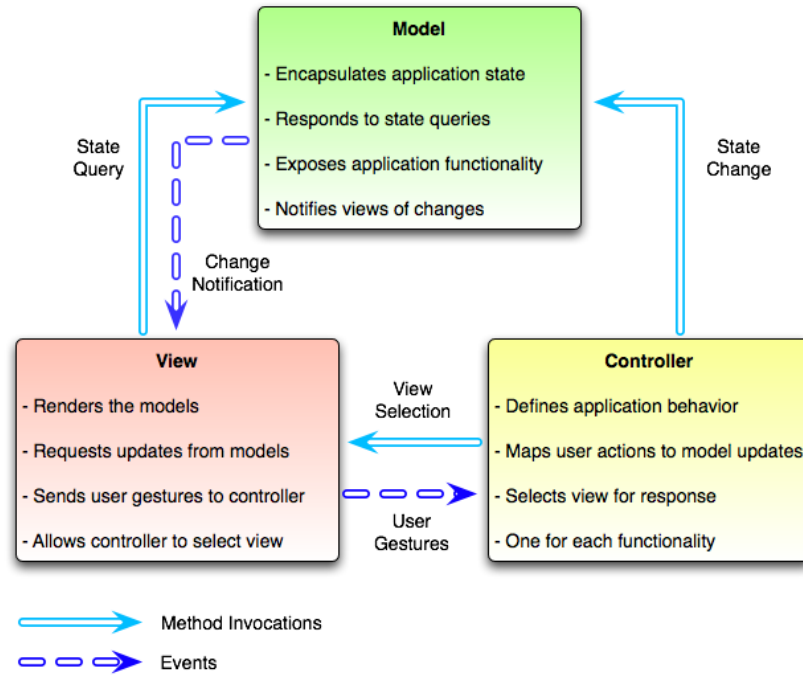


Figure 8: Representation of the MVC structure.

The information in this chapter allowed to understand some of the technologies and the protocols used, in this dissertation. It was explained how does the SIP works, and how it can be related to HTTP.

Other important information on these chapter is the presention of a basic IMS architecture and, in particular, the IMS architecture of PT Inovação. Finally, another two concepts, with relevance to this work, were presented. These concepts were the XCON Standard and the MVC Pattern.



## 3 Web Communication Services

In the context of this Thesis two independent services called *Tagarela* and *WebHuddle* were analysed. The service *Tagarela* is a solution in development at PT Inovação for conference and collaboration, while *WebHuddle* is an *open source* service for web conferencing as well.

One of the main purposes of this dissertation is to integrate some *WebHuddle* and *Tagarela* features in a single service.

In the following sections these two services will be introduced, as well as the features of each one. It will also be explained their architectures and the entities available to interact with these services.

### 3.1 Tagarela

*Tagarela* is the Conference and Collaboration solution that is under development in PT Inovação, which is part of an architecture called *Shipnet*. This architecture includes the family of products to give answer to challenges and needs of the NGN, in the FMC scenario.

*Tagarela* provides a wide range of features. These features include chat, video conferencing, file sharing and audio conversation. Under this Thesis, two new features were added: desktop sharing and slideshow sharing. These features were implemented using the *WebHuddle* (it will be presented in section 3.2). To better understand how does the *Tagarela* works, there are some concepts that need to be explained [26].

- **Focus:** Is an SIP UA that is represented by the conference URI, and identifies the session. The focus keeps the signalization with each one of the conference participants, and it is responsible to ensure that each participant receives the media that makes part of the conference session;
- **Floor:** Temporary permission to control some shared resources, inside one session;
- **Floor Chair or Moderator:** Logical entity that controls a specific floor (gives, denies or repeals the floor). The entity that takes the place of Floor Chair can, at the same time, take another role, like for example, it can be one participant;
- **Floor Control:** Mechanism which provides, to the applications and users, the possibility to access and change, simultaneously, the conference object;
- **Floor Control Server:** Logic entity that keeps the state of the floors, including who has the floor (Floor Chair), which floor users have, etc. The requests to manipulate the floor are sent directly to the Floor Control Server.

#### 3.1.1 Architecture

The Conference and Collaboration solution, *Tagarela*, adheres to the 3GPP IMS and ETSI Telecoms & Internet converged Services & Protocols for Advanced Networks (TISPAN) architecture. In particular it

meets conference standards 3GPP TS 24.147 ([27]) and the XCON IETF standard ([28] [29] [30] [31]).

Thus, the PT Inovação conference solution is one application executed in the functional element SIP AS, which controls the sharing of the resources used at the conference sessions, made available by the functional elements Multimedia Resource Function (MRF). A representation of the main functional architecture of *Tagarela* can be seen at figure 9. It is important to note that this architecture is just a conceptual architecture.

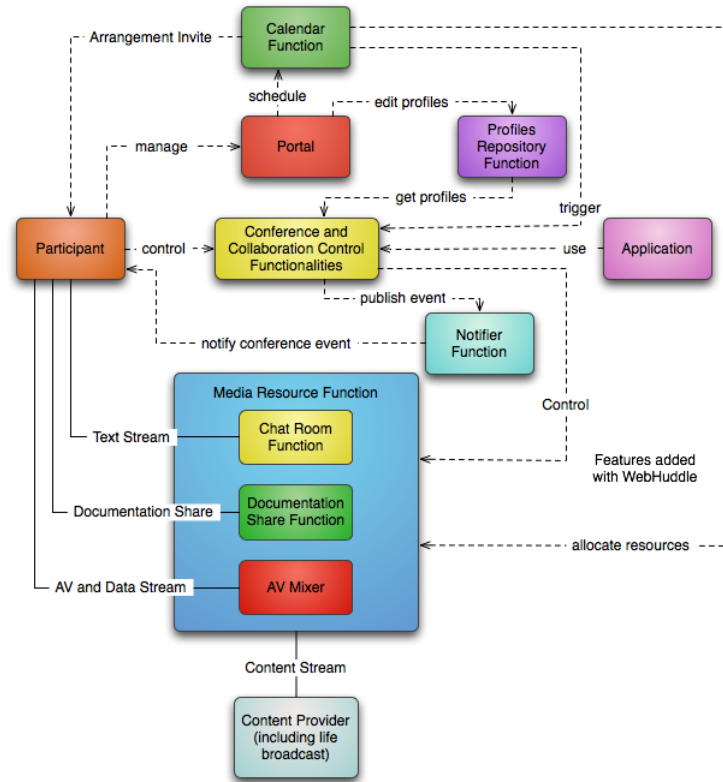


Figure 9: Functional Architecture of *Tagarela*, before integration with *WebHuddle*.

- **Calendar Function:** it allows to manage the previous scheduling of the *Tagarela* sessions, by sending invitations to the participants, while ensuring the availability of the sharing resources on the sessions. The calendar service is also responsible for sending reminders to the scheduled sessions, and it may trigger the establishment of the session (Conference and Collaboration Control Function (CCCF)). This functionality can be provided by an extern calendar server compatible with the standards iCAL [32];
- **Portal:** provides the administrators with the necessary tools to create and manage the new *Tagarela* services classes, through the definitions of the sessions profiles and respective control policies. The Portal also makes available *self-provisioning* functionalities, allowing users to manage their own



subscriptions and profiles (using the Profile Repository), as well the scheduling of new sessions, through the Calendar Function;

- **CCCF:** entity responsible for the logic that defines the final behavior of the *Tagarela* services. CCCF provides the user with the functionalities of Floor Control. This is made through different user interfaces and to control the several resources shared on the *Tagarela* sessions (Resources Share Management (RSM)). With the RSM the user can control the participants audio and video streaming (MRF), chat sessions (Chat Room Function) and the documents sharing (Documentation Share Function), including Word, PPT and Excel files.

The events that are generated by the session control (e.g., entry of new participant) are published and reported by the presence service, in accordance with the received subscriptions from participants (or others applications) (Presence Function).

The sessions are governed by policies (rules) that are in the service and users profiles repository (Profiles Repository Function). The session establishment can be triggered by received triggers of the calendar service (Calendar Function), to the previously scheduled sessions at the Portal;

- **Profiles Repository Function:** implements the functionalities of Conference Policy Server and it is responsible for the services profiles management and its users, providing reading and writing functionalities to the Portal, and just reading functionality to the CCCF;
- **Notifier Function:** provides the required features for the reporting of events related with the *Tagarela* session. These features are provided through one normalized event package, specific of the conference service (Conference Event Package) [30];
- **MRF:** provides the control functionalities to the sharing of several resources, including:
  - **Chat Room Function:** functionality of group service that is part of the MRF functionalities, where, in this case, the media is the text. Allows to control the exchange of instant messages by group of users;
  - **Documentation Share Function:** another group service functionality, that allows to control file transfers (ability to upload files to one central server, to which the other conference participants have access, and from which they can do the download of the files), and the sharing of viewing and editing of different types of documents;
  - **Audio and Video (AV) Mixer:** mix and distribution, in real time, of the audio and video media flows, by the group of participants in the *Tagarela* sessions. This feature is provided by the Media IP-Windless server.
- **Content Provider:** extern entity that provides contents to the MRF, in real time, to share with the participants of the *Tagarela* session.

### 3.1.2 Main Features

*Tagarela* has several features available, but only some will be detailed here. The features that will be presented are the required ones to understand how *Tagarela* works, and the main functionalities available to the user. There are two types of conferences that can be created using *Tagarela*: Thematic and Ad hoc.

- **Thematic Conferences:** a conference of this type can only be created by the Administrator of *Tagarela*, and they only are visible to an user after he has subscribed the room. To access to one thematic conference the user has one set of processes that can be used. One of them is the prior knowledge of the conference address, and so the user may enter the room just with one call to that address.

Another way to access a thematic conference is through the consult of the service page, where there is the indication of all ongoing sessions, and then the user may request the service to invite him. At last, the user can be invited by one participant of that session. These accesses to the conferences can be controlled by sessions subscription or by the number of participants allowed;

- **Ad hoc Conferences:** this type of conferences can be created by any user. The beginning of one ad hoc conference may be done through the web interface of the service. On this web interface are the addresses of the users, that the creator may invite.

The number of participants, in each conference, is limited up to a maximum number, defined at the time of the conference creation. To enter in one Ad hoc room, to which the user was not invited at the begin, the user needs to be invited by a participant of that meeting.

These types of conferences (thematic and ad hoc) can also be divide into two, depending on the type of media supported. They can be audio conferences or video conferences.

- **Audio Conference:** *Tagarela* allows users to enter in audio conferences sessions, where the users share their audio, using VoIP and PSTN terminals. These conferences can, in any moment, support video and chat between participants. The control of this support is responsibility of the business logic;
- **Video Conference:** The users of this service can participate in video conferences, so they can share their video resources among themselves. It will be created one single video stream, with a mosaic of all participants, to be watched in a client with video support. As other conferences, the users can also share the audio and the chat features, while participating in the session.

Once inside a conference, whether it is thematic or ad hoc, there are a set of features that the participants can use. Some of these features are only available for some type of participants (Administrator or Moderator). These features are presented and explained below:

- **Add participant:** This feature allows to invite new participants to the ongoing conference. Depending on the conference rules, this can be done by the moderator or by any participant;
- **Remove participant:** The moderator has permissions to remove participants that are disturbing the room;
- **Banish participant:** If one specific participant is constantly disturbing the conference, the moderator may decide to banish him. With this feature, the banished user is removed from the room and he no longer is allowed to enter the room, until the moderator wants to. This action can only be made by the moderator of the conference room;
- **Mute/unmute audio:** Allows to *mute* or *unmute* one participant, so that the audio sent by that participant ceases to be listen by the remaining participants;
- **Mute/unmute video:** This feature works just like the audio mute/unmute, but with the difference that instead of audio, this feature allows to mute/unmute the participant video. This means that there will be no more available streaming from the participant;
- **Mute/Unmute all but himself:** Allows the moderator to silence all the participants except himself;
- **Delegation of moderation:** The participant with privileges of moderation can choose to delegate these privileges to other participant;
- **Close/Open conference:** At any time it can be defined that is no longer allowed the entry of more participants in the conference (Close Conference). In the same way, the conference can be opened to new entries at any time (Open Conference).

### 3.1.3 Entities

In *Tagarela* there are different types of users, also called entities, and each one of them has different roles in the service. They are organized hierarchically, being the Administrator the entity with more privileges.

- **Administrator:** Entity that is responsible for managing the service. The Administrator is also responsible for the creation of the *Tagarela* services classes, and by the definition of the rules of use (type of shared resources, maximum number of participants, moderator, etc);
- **Session Creator:** Entity responsible for the creation of the *Tagarela* session (thematics and/or ad hoc conference) and definition of the session policies and respective mode of control, according to the privileges given by the Administrator.

Although the Session Creator is not necessarily a participant of the session created by him, he has the full control of the session. The only entity superior to the Session Creator is the Administrator of the service;

- **Moderator:** It is the entity responsible for, in real time, management the conferences. During a conference, the moderator has the ability to give, revoke and/or deny the ability to use the room resources (file sharing, video sharing, chat, download files that are being shared, etc), to participants. The moderator can also remove a participant from the conference.

The choice of the moderator and his moderation functions are defined according to the policies defined by the Session Creator. The moderator can also be chosen during the conference creation;

- **Participant:** This entity represents the user without moderation privileges, when he is in a conference room. Once the user is no longer in the meeting, it will become default to a simple user. The functionalities of participant control are defined by the Session Creator;
- **User:** It is the entity that executes the actions outside the sessions. For example, the user is who does the subscription and scheduling of an *ad-hoc* meeting.

### 3.1.4 Web Interface

*Tagarela* has one web interface available, based on Asynchronous JavaScript and XML (AJAX) technology, so that it can improve the user interaction with the service. In figure 10 is represented the login page of *Tagarela*. Through this page, the user performs his logon on the service, and he has the possibility of choosing the language that he wants to use. The options currently available are Portuguese and English.



Figure 10: Login page of *Tagarela*.

## 3.2 WebHuddle

*WebHuddle* is a multi-platform *open source* application, based on a web server, for web and video conferencing [33]. The goal of analysing this application is to add two new features into a Conference Service that is under development at PT Inovação, called *Tagarela*. The features that were added to the Conference Service are the desktop and slideshow sharing. With these features the user will be able to share their desktop, pictures and presentations, with the users who are at the meetings.

### Key Features:

- **Simple:** The client runs the service in its browser, through firewalls and proxies, and requires no installation. Furthermore, it has an intuitive user interface;
- **Small:** The client applet only needs between 75 to 175 kB of disk space, depending on the platform and the used features;
- **Standard:** *WebHuddle* works in Linux, Windows, Unix and Mac OS, through Java technology. It also uses the same protocol as browsers, HTTP;
- **Open Source:** Since it is an *open source* application, *WebHuddle* has the advantages of being transparent and flexible.

### Limitations:

- **Type of uploaded files:** Unfortunately, the format of files supported is limited. For example, it does not support *doc* or *pdf* files, which are two popular format used in documentation exchange;
- **Upload before sharing:** *WebHuddle* does not support file upload during a slideshow sharing, so the files must be uploaded before beginning the shared session;
- **View shared area:** In a desktop sharing, when is being shared just one area of the desktop, the user does not have a precise idea of what he/she is sharing.

### 3.2.1 Architecture

The *WebHuddle* architecture is composed by a Server that publishes its services on the Internet, through JBoss [34], in a configurable URL, and the users who access to available services. But the *WebHuddle* Server himself acts as a client and a server. This is, the client is the entity that receives the requests from the browser, and sends these requests to the server entity. Another important feature of *WebHuddle* is that it is based on the MVC pattern (defined in section 2.4.3), because the *jsp* pages are used to render the *View*, the Servlet works as *Controller* and components Enterprise JavaBeans (EJB) act as the *Model*.

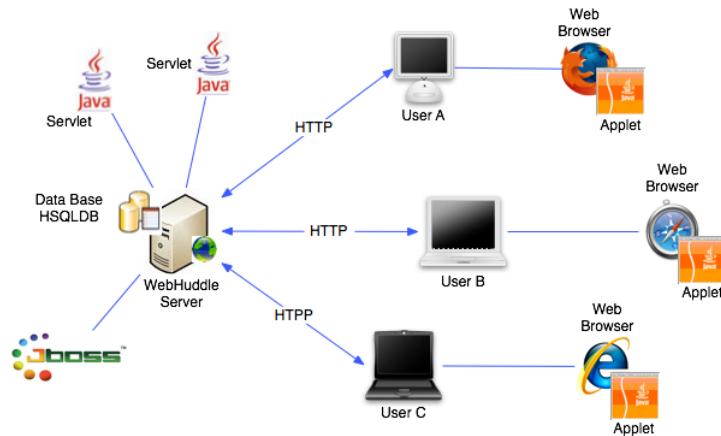


Figure 11: Architecture of *WebHuddle*.

The *WebHuddle* Server is the entity responsible for the management of all the interactions between users. Furthermore, as the events are received the server has to perform the required actions. With regard to the information, this is stored in a Hypersonic SQL (HSQLDB) database, and it has all the information about the registered users. For example, the pictures that the users have on their accounts, the meetings that they have attended and the contents (users and pictures) associated to each meeting. This information is stored in a file at the WebHuddle Server.

To make the service available on the Internet, it is required that the service has been deployed at JBoss, which then should be started. JBoss is an *open source* application server, based on J2EE platform. It is implemented in Java and, because of this, it can be used in any Operating System (OS) that supports Java, just as what happens with *WebHuddle*. When the JBoss service is started, the *WebHuddle* will be available on the Internet, at the *WebHuddle* Server IP address. When the service is published on the Internet, a group of servlets, on the server side, will be available and, on the client side, there are a group of applets that are loaded on the client browser. A representation of the *WebHuddle* architecture is depicted in figure 11.

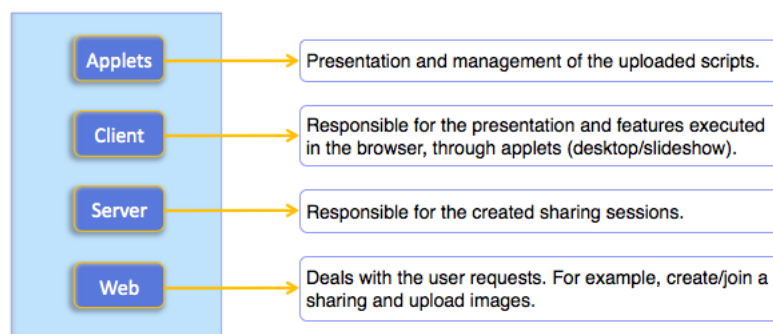


Figure 12: Principal components of structure of the *WebHuddle* framework.

The framework of *WebHuddle* is structured in several folders, and the main are: "applets", "client", "server" and "web" (figure 12). Despite what it may seem, the "applets" folder only has the code responsible for the applet used to manage the slides, at the Server. The others applets that *WebHuddle* use are inside the "client" folder, maybe because these applets runs on the client side.

At this stage, it is important to explain the difference between *script* and *slide*. These two concepts are only different when the uploaded file is a *zip* or a presentation. In this case, the *script* corresponds to the file uploaded and *slide* is each image of the *zip* or each page of the presentation. In the case that the uploaded file is only one image (*bmp, jpg or gif*), the *script* has only one *slide*.

Summarizing, the *script* is composed by one or more *slides*. So, when a slideshow sharing is started, the content that is shared corresponds to a *script*, whose identifier is inserted in the URL that is used to start the sharing (example of the URL is presented in section 3.2.6).

To perform the interface between the *WebHuddle* Server and the database, EJBs are used. When the service is deployed it creates the database tables (see Appendix B which contains the tables). These EJBs are inside the "server" folder, represented in figure 12, and the main ones are the following:

- **CustomerBean:** stores information about, for example, the user name and e-mail, user IP and timestamp of the account creation;
- **LogonBean:** contain information about the logon Identifier (ID), the IP from where the user performs the login, the user agent (Internet Explorer (IE), Firefox, etc) and the time of the login;
- **MeetingBean:** stores information about, for example, the meeting ID, meeting name, date and time of start and end of the meeting;
- **ParticipantBean:** EJB responsible for storing information about the participation of an user in a meeting. The information stored is, for example, logon name and e-mail of the user who entered the meeting, java version, user IP and start and end time of the participation;
- **ScriptBean:** contains information about the script ID and name;
- **ContentslidBean:** has information about the size and name of the slides.

### 3.2.2 Struts

*Struts* is an *open source* framework for implementing applications using the MVC design pattern. This allows developers to concentrate on the business logic, without worrying about the others parts of the architecture [35].

The *Controller* is provided by *Struts* in the form of an *ActionServlet* class. This class is used for handling all requests. In order to *WebHuddle* to allow the use of this servlet controller, the code represented in figure 13 is included in the deployment descriptor ("web.xml"). In file "web.xml", it must also

be specified the mapping of *WebHuddle* URLs to this servlet. The code of figure 13 also defines that any URL with ".do" extension calls the *ActionServlet* to handle the request.

```
<servlet>
  <servlet-name>action</servlet-name>
  <display-name>action</display-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>/*.do</URL-pattern>
</servlet-mapping>
```

Figure 13: Example of code that allows to enable a controller servlet.

To carry out the required actions when a particular request is received, the *ActionServlet* need to be configured using *Action Mappings* in the "struts-config.xml" file. An *Action Mapping* specifies a fully qualified Action class name, that should be invoked when an URL matching its path is requested. The example of an *Action Mapping*, when the URL `http://WebhuddleServerDomain:8080/desktop.do?username=ConferenceRoom&meetingName=ConferenceRoom` is requested, is in figure 14.

```
<action path = "/desktop"
  type = "com.sts.webmeet.web.OpenMeetingAction"
  name = "openMeetingForm"
  scope = "request"
  input = "meeting" >
  <forward name = "success" path = "/participantapplet_desktop.jsp" />
</action>
```

Figure 14: Example of an action defined in *struts-config.xml*.

When an URL for the *WebHuddle* Server with name "desktop.do" is called, the *ActionServlet* looks for a mapping with the *path* "/desktop". It then instantiates the class defined by the *type* attribute ("*com.sts.webmeet.com.OpenMeetingAction*") to perform the required action. Any form parameters in the requested object are passed to the Action class, using the form bean defined by the *name* attribute ("*openMeetingForm*"). This attribute defines a logical name for the form bean, and its mapping is represented in figure 15. Using this input, the Action class executes operations to serve the request. Depending on the result of the operation, the user is forwarded to the next view, using the *ActionForward* object (see tag *forward* used in figure 14).



```

<form-bean name = "openMeetingForm"
           type = "com.sts.webmeet.web.OpenMeetingForm" />

```

Figure 15: Example of a form bean defined in *struts-config.xml*.

### Definition of Action Class

Is an adapter between an incoming HTTP request and the corresponding business logic, that should be executed to process this request. It should define a method that will be automatically invoked when a request is received. For each request received, the *Controller* (ActionServlet class) will select an appropriate Action class, it will create an instance and it will call the perform method. This method should be implemented to execute the business logic for serving the request, handle error and navigate to the appropriate view based on the outcome of the action. Usually, the business logic part of the implementation is delegated to EJB.

### Action Mapping Configuration

For mapping a request from an URL to a particular Action class, the ActionServlet class uses the ActionMapping interface, in order to be provided with suitable information. *Struts* provides a declarative way, in the form of XML configuration file, to pass this information to the controller servlet. Then, this file is parsed by the controller servlet, which in turn gets the required information to handle the incoming request.

The configuration file, "struts-config.xml", was defined and placed in the "WEB-INF" folder of the *WebHuddle* WAR file. The controller servlet knows the configuration file from the "web.xml", where it is declared as represented in figure 16.

```

<servlet>
....
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
</servlet>

```

Figure 16: Definition of the location of the *struts-config.xml*.

The "struts-config.xml" is a configuration file with a well-defined structure. Its important elements are the following:

- **form-beans:** contains the definition of the beans used by the application. A *form-bean* describes an instance of ActionForm class that will be used to pass the form data to the action class;

- **global-forwards:** defines the forwards that can be used by all action mappings. A *forward* is an instance of ActionForward class that maps a resource name to the actual resource in the application. These forward names are used in the application to forward the control to the next view, based on the outcome of the request execution;
- **action-mappings:** element used for defining mapping for various actions using *action* tags. This tag is used to map a request URL to a particular action, and to pass the associated information.

### 3.2.3 Entities

In *WebHuddle* there are only two entities that interact with the application. They are:

- **Moderator:** It is the entity that starts a desktop sharing or a slideshow sharing. It controls what the other users will see, during the sharing:
  - In one **desktop sharing** he is responsible for deciding when to start and to stop the sharing. The moderator may also decide if wants to share all desktop or just a single area, bounded by a rectangle;
  - In one **slideshow sharing** he is responsible for choosing the slides that wants to share, and the order of the presentation of the slides. The moderator also has the ability to do some kind of edition of the images, through the tools described in the previous section.
- **User:** Entity that joins into one desktop or slideshow sharing. It has no control on what is happening during the share.

### 3.2.4 Features

Besides desktop and slideshow sharing, *WebHuddle* has a set of services, including chat, VoIP and meeting recording. However, these services will not be presented, since they are outside the scope of this thesis. To interact with *WebHuddle* is necessary to have a registered account and to be logged in. A *WebHuddle* user will correspond to a meeting in *Tagarela*. With this, for example, uploaded files in *WebHuddle* will correspond to files that an user wants to present in a meeting at *Tagarela*. The *WebHuddle* features that are used are the following:

- **Register:** allows the registration of a new user;
- **Login:** enables a registered user to login into a *WebHuddle* session;
- **Slides upload:** provides an authenticated user to upload image(s), called slide(s). The slides are added to the user account, and become available for when the user wants to start a slideshow sharing session;

- **Slides manage:** allows an authenticated user to manage the slides that will be associated to the account. The users can select, from all slides of his account, which will be available in the slideshow sharing;
- **Desktop sharing:** allows the coordinator user (*moderator*) to share his desktop with users that are at the meeting. The moderator can choose if he wants to share his entire desktop or only a part of it, and can also control the start and the end of the desktop sharing. To desktop sharing two available actions exist:
  - **start:** an authenticated user can start a desktop sharing. When the user starts a sharing he/she becomes the session *moderator*;
  - **join:** an user, authenticated or not, can join a sharing session;
- **Slideshow sharing:** allows the user who starts the meeting (*moderator*), to share a set of pictures (slides), with the users that are in that meeting. When the moderator shares more than one slide, the users only see the slide that the moderator is showing. In a slideshow sharing the moderator has a set of tools to mark something on the slides. These tools are: *undo* (undo the last change), *erase all* (erase all the changes made during the slideshow sharing), *change color* (allows to change the color), *rectangle* (draw a rectangle area on the image), *line* (draw a line) and *scribble* (write or draw something on the image). At figure 17 can be seen these buttons and their description.

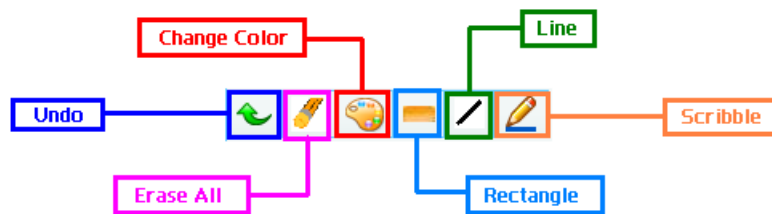


Figure 17: Description of the buttons that the *moderator* can use during a slideshow sharing.

### 3.2.5 Work With WebHuddle

There are a set of actions that are used by *WebHuddle*. However some changes were made to the way how *WebHuddle* works, so that the integration with *Tagarela* could be made. Next, it will be explained these set of actions, and the changes that were made. At figure 18 it can be seen how is the initial interface of *WebHuddle*, after user do the login.



Figure 18: Interface of the *WebHuddle* Server, at the beginning.

### 3.2.5.1 Standard Features

In this section it will be explained the features available in *WebHuddle*, before the changes made to integrate it with *Tagarela*.



- **Register:** If an user wants to register in the service, he must indicate the first and last name, e-mail address and country initials. The supported countries are Portugal (pt), Spain (es), England (en), France (fr), Germany (de), Finland (fi), Netherlands (nl), Slovakia (sk) and Hungary (hu).

The field *country initials* allows all the *WebHuddle* features to be written in the respective language. When the user ends the register, *WebHuddle* generates a random password, that is sent to the user e-mail. Then the user can change the password in the "Profile" menu;

- **Login:** When a user wants to login in the service, he has to enter the e-mail and password;
- **Profile:** Feature that allows an authenticated user to change some values of his/her account. These values are the user first and last name, password and e-mail address;
- **Upload:** When an user wants to perform a slide upload, he/she has to write the slide name and browse the image that wants to upload. The user also needs to define the size that wants to assign to the image. The size can be 320x240, 640x480, 800x600 or 1024x768.

The uploaded files can only be one of the next types: gif, jpg, bmp or zip (containing those types). *WebHuddle* also allows the upload of presentation with ppt or sxi extensions. But to do this, the *WebHuddle* Server need to have an *OpenOffice* instance running, because *WebHuddle* uses this application during the upload of the presentations, to convert each page into an image;

- **Begin Meeting:** Allows a logged user to create a new meeting. This feature has a set of fields to be filled by the user. These are: meeting name, meeting description, slides to share, meeting password and options to choose the recording and VoIP features.

Is also worth noting that the meetings have the two services embedded (desktop and slideshow sharing). The *moderator* can choose, during the meeting, which service wants to use, through the buttons  and , respectively;

- **Join Meeting:** This feature allows an user, authenticated or not, to join a meeting. *WebHuddle* needs the next four fields, to join an user into a meeting: e-mail address and name of the user that wants to join, e-mail address of the moderator and the meeting password (defined during the meeting creation);
- **Schedule:** Allows to schedule meetings, defining day and time to start the meeting;
- **Recording:** Feature that allows the moderator to choose if he wants to record, or not, the meeting. This is set when the user is creating the meeting.

### 3.2.5.2 Modified Features

In this section it will be explained the features available in *WebHuddle*, that were modified after the integration with *Tagarela*.

- **Register:** The register function is used when a meeting is created in *Tagarela*. When a meeting is created, it is sent an URL to the *WebHuddle* Server, to do the login, with the new account username. The Server verifies that this account does not exist, and creates it. There is no need to insert a password, because instead of generating a password to the new user, the account password is equal to the username (in this case, the *Tagarela* meeting name);
- **Login:** To login the user only has to enter the username. It is important to note that the username is the value assigned to the e-mail field;
- **Profile:** This features allows to change the account name. When, in *Tagarela*, a conference name is changed, it is sent a request to *WebHuddle*, in order to perform the change on the account name;
- **Upload:** The user only need to browse the image to upload. The name of the slide will be the uploaded file name, and the scale is pre-defined to 800x600;
- **Begin Meeting:** Now to create a meeting, while in a conference room of *Tagarela*, is called an URL of *WebHuddle* API, with the type of meeting (desktop or slideshow sharing), the meeting name and the *WebHuddle*'s username. To identify the *moderator* of the created meeting in *WebHuddle*, the value of the meeting name and the username is the name of the conference room of *Tagarela*;
- **Join Meeting:** After the integration with *Tagarela*, when an user wants to join a meeting, is called an URL of *WebHuddle* API. This URL has the type of sharing that wants to join, the meeting name, the name of the conference room and the username of the user that wants to join the meeting;
- **Schedule:** This feature was removed after the integration with *Tagarela*, because this service was already available;

- **Recording:** Since this feature was not on the scope of this Thesis, it was removed from *WebHuddle*.

### 3.2.6 Current WebHuddle API

The *WebHuddle* API is used to create meetings (desktop and/or slideshow sharing), to add participants to it and it also allows to manage slides for using at the slideshow sharing.

Conceptually, a meeting belongs to a logged user (moderator) and each meeting has one or more participants. To create a meeting, the API needs the type of meeting (desktop or slideshow), the username and the meeting name. After an user is authenticated it is possible to create a meeting he/she only has to set the meeting name and username.

The descriptions of the URLs that are used by the *WebHuddle* Server API are the following:

- **Login and Registration:** the difference between the *login* and *registration* is that when *WebHuddle* receives a login username of an account that does not exist, it creates it.

- `http://WebhuddleServerDomain:8080/logon.do?username=Room`

- **Desktop Sharing:**

- `http://WebHuddleServerDomain:8080/desktop.do?meetingName=Room&username=Room`

- **Slideshow Sharing:**

- `http://WebHuddleServerDomain:8080/slideshow.do?meetingName=Room&username=Room&scriptID=Number`

- **Join Desktop Sharing:**

- `http://WebHuddleServerDomain:8080/enterroomDesktop.do?action=enter&meetingName=Room&hostEmail=Room&participantName=User`

- **Join Slideshow Sharing:**

- `http://WebHuddleServerDomain:8080/enterroomSlideshow.do?action=enter&meetingName=Room&hostEmail=Room&participantName=User`

Because the list above is composed by examples, some parameters must be explained:

- **WebhuddleServerDomain:** it must be replaced by the *WebHuddle* Server IP;
- **Room:** corresponds to a *Tagarela* room name. For *WebHuddle* Server this *Room* parameter is one user account;

- **User:** is the name of the participant that wants to join a sharing. This user does not need to be registered in *WebHuddle*, because in *WebHuddle* only are registered the *Tagarela* conference rooms;
- **Number:** is the value assigned to *scriptID*, and represents the script that the user wants to share in the slideshow sharing that he/she is going to start.

### 3.2.7 WebHuddle and Network Address Translation (NAT)

A very important benefit of *WebHuddle* is that it works through Network Address Translation (NAT) [36]. To best understand what NAT is, a brief definition is presented in the following. NAT enables private networks addresses to access public IP addresses through a translation service. Most of the NAT configurations maps all of the private IP address, on a home network, to just one IP address. This allows all the computers on a Local Area Network (LAN) to share a single Internet connection. Another advantage of NAT is that it enhances LAN network security by limiting the access of external computers into the home IP network space.

NAT acts by snooping the incoming and outgoing IP datagrams on the Internet, and if it is needed it modifies the source or the destination address, respectively, in the IP header (and the affected checksums) to reflect the configured address mapping. NAT is usually used on router and other gateway devices at the LAN boundary. Since all the interaction made with *WebHuddle* Server are made through HTTP, using the port 8080, this service works through firewalls.

### 3.2.8 Capturing Image for Desktop Sharing

When the desktop sharing *applet* is loaded, there are a set of functions that are called by it. The desktop applet, "*AWTClientDesktop.java*", creates a new instance of the "ContentManager" object. This object is responsible for the invocation of all the necessary functions, to interact with the applications, and to associate this functions to buttons that are presented in the applet interface.

During the creation of the applet, it is also called an object that will be responsible for doing the desktop sharing interface, "*AppView*". In this object there are functions that will detect the area that the moderator wants to share (all desktop or just a specific area). It also detects the events during the sharing, such as the resize of the rectangle and the in and out of the mouse from the sharing area.

When the "Play" button of the *applet* interface, is clicked a thread is launched to start the desktop sharing. This thread receives the data that is being shared, and sends this data to the Server, so it can forward the data to the participants of the sharing. This thread is created in the file "*AbstractScreen-Scraper.java*". In this file is the function responsible for capturing the image to share, that is called "*doScrape()*". This function is being executed until the moderator clicks on the "Stop" button.

During the execution of the "*doScrape()*" function, is made a "sleep" on the thread during a time, that is specified in a *WebHuddle* properties file (E.g., *webhuddle.property.appshare.pixels.per.millisecond=240*).

This time is used for controlling the bandwidth use. For example, if the moderator is sharing a desktop that is 800x600, the screen scrape will be done every two seconds ( $800 \times 600 / 240 = 2000$  milliseconds).

Within this chapter the web communication services used, in this dissertation, were presented. It was explained the architecture, the features available and how these services can be used.

For example, for *Tagarela* the existing entities were presented. Concerning to *WebHuddle*, it was explained how the service receives the HTTP request and interprets them, in order to trigger the required actions.



## 4 Implementation

### 4.1 WebHuddle and Tagarela Integration

In this section it will be presented how the new features were added in the *Tagarela* service. At figure 9, of section 3.1.1, it was presented the *Tagarela* architecture before adding the desktop and slideshow sharing, through *WebHuddle*. In figure 19 the changes made, in *Tagarela* architecture are shown, after the integration process. In this figure it can be seen two new components of the architecture. These components will be presented in the following.

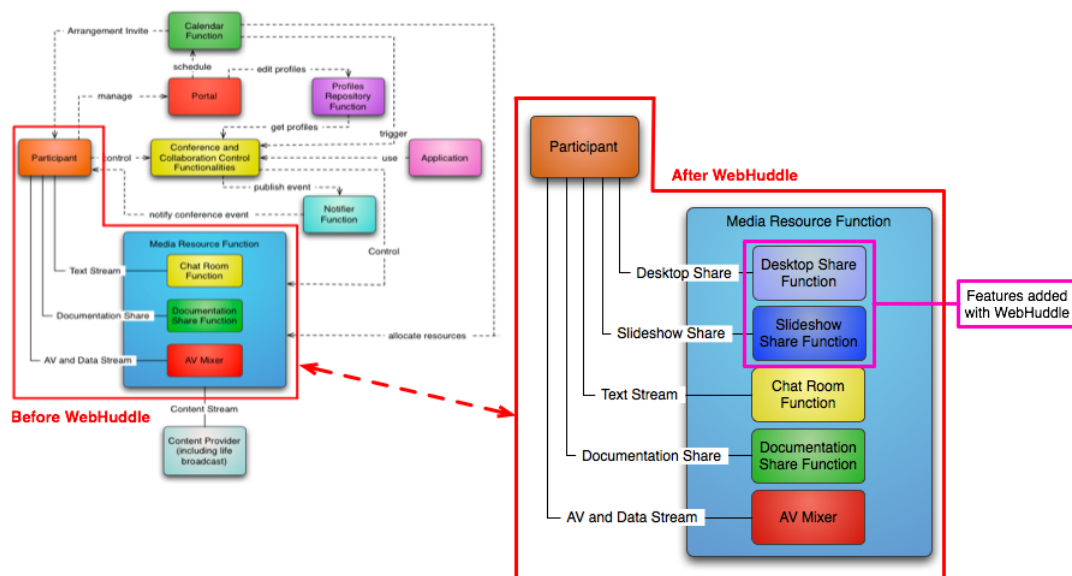


Figure 19: Changes to the functional architecture of *Tagarela*, after the integration of the *WebHuddle* features (desktop sharing and slideshow sharing).

- Desktop Share Function:** this new service group functionality allows to control the desktop sharing. It provides the ability to show, to others participants, what is happening on the desktop of the user that started this type of sharing. This does not allow others participants to access the desktop that is being shared;
- Slideshow Share Function:** this new group service functionality allows the users to share and control a set of pre-uploaded pictures and presentations.

It is important to make a distinction between two concepts. Both in *Tagarela* and in *WebHuddle*, there is an entity called *Moderator*. After the integration, the *Moderator* of *Tagarela* (from now on designated as *ModeratorConf*) may not correspond to the same as the *WebHuddle*'s (from now on designated as *ModeratorWH*). In brief, the *ModeratorConf* is the responsible for managing the conference room of

*Tagarela*, and the *ModeratorWH* is the participant of *Tagarela* who starts the desktop or the slideshow sharing.

The entities *ModeratorWH* and *ModeratorConf* do not need to be the same entity, because the desktop sharing or slideshow sharing do not need to be started by the *moderator* of a conference room. It means that a participant that is not the moderator may want to start a sharing, if he/she has permission to do that.

#### 4.1.1 Create New Conference

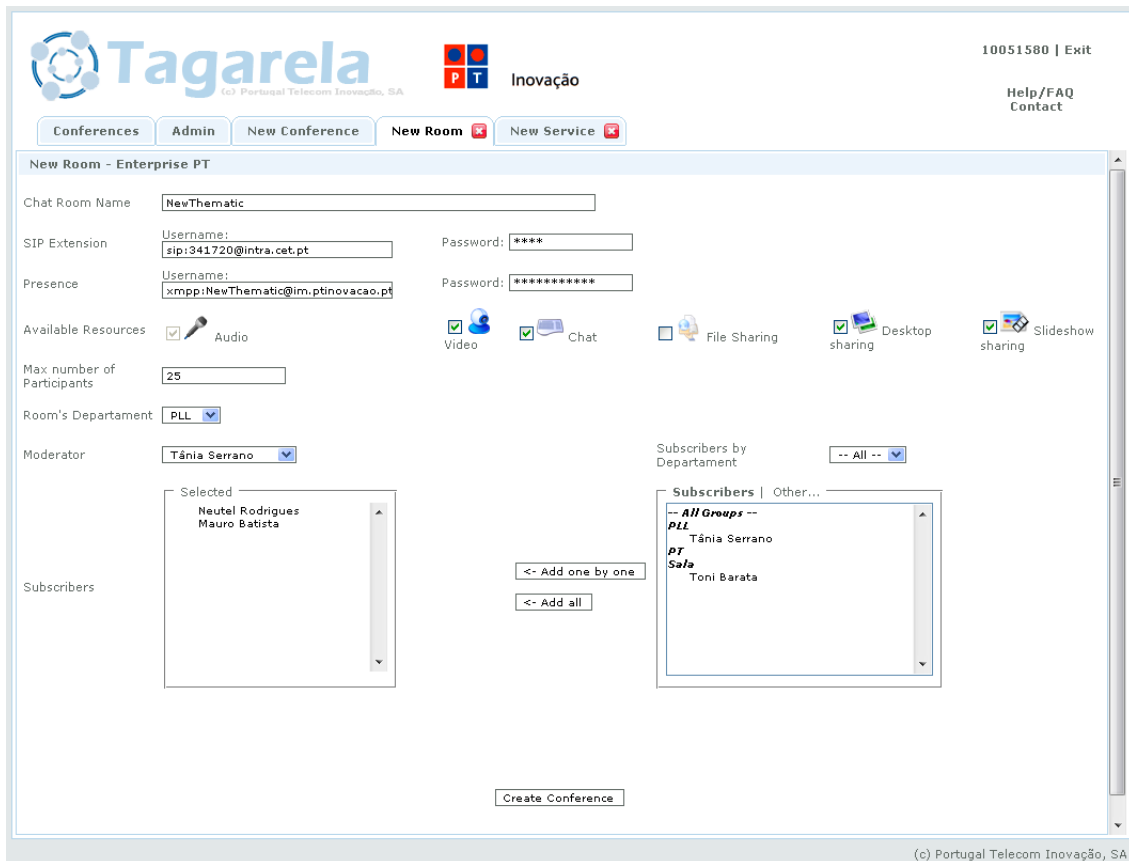


Figure 20: Interface to create new thematic conference.

In the *Tagarela* interface (see figure 20), when an user creates a new conference room (Thematic or Ad hoc), it will be also created a new account in *WebHuddle*. This means that one new *Tagarela*'s room corresponds to a new *WebHuddle*'s account. This works this way because the slides that the users want to show, in one conference room, must have been uploaded to one account at *WebHuddle*. Summarizing, a room called "ThematicRoom" will use a *WebHuddle*'s account with the same name. If the Session Creator creates one new ad hoc conference, (see Appendix C for a representation of the interface), the new *WebHuddle*'s account is created the same way.

After the integration with *WebHuddle* it became possible to create one conference with the desktop and/or slideshow sharing resources. It is possible to choose this type of resources the same way as the other resources are chosen. Every time that is created one new conference, it is also created one account in *WebHuddle* Server, with the name of the conference. The request to create a new *WebHuddle*'s account is made through an URL, as for example:

- <http://WebhuddleServerDomain:8080/logon.do?username=NewRoom>

When this URL is received by *WebHuddle* Server, it gets the username and, because this account is not registered, it creates the new account. At figure 21 is the sequence diagram for the creation of a new conference room, showing the interaction between the *Webhuddle* Server and *Tagarela*.

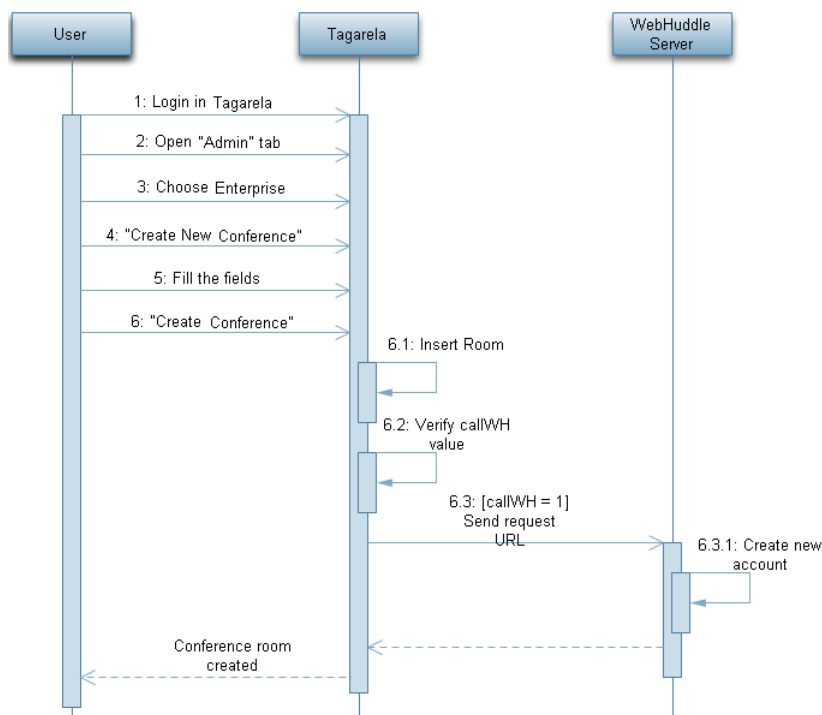


Figure 21: Sequence diagram of the creation of a new conference room.

As it can be seen in figure 21, there are a set of steps that are needed, before an user is able to create a new conference room. If the user is not an administrator, the user is not allowed to create a thematic conference, but he/she can create a new ad hoc conference room.

First the user need to login in *Tagarela*, and if he/she is administrator of the system, it is needed to click in the "Admin" tab, of the interface. After this step the user needs to choose the Enterprise to which he/she wants to add the new thematic conference, and must click in the "Create New Conference" button.

In the next stage of the creation of a new conference, the user need to fill all the fields of the setup page (see figure 20), and next he/she needs to click in the "Create Conference" button, at the bottom of the page. When *Tagarela* receives the new conference event, if all fields are well filled, the new room is inserted in the database.

In the case that the conference room is created successfully, by *Tagarela*, a *flag* is assigned with a specific value. At the end of the creation process, the *flag* value is verified. If the *flag* has the wanted value, a request of login is sent to *WebHuddle*, with the new conference name as parameter. Because *WebHuddle* notes that this account is not registered, it creates a new account with the room name, obtained from the request URL. This is the last step to create a new conference room.

#### 4.1.2 Joining a Conference

If one user wants to join in one conference room, he/she must choose the conference to join. After choosing the conference room, the user must provides the SIP account that wants to use, in order to join the conference room (see figure 22).

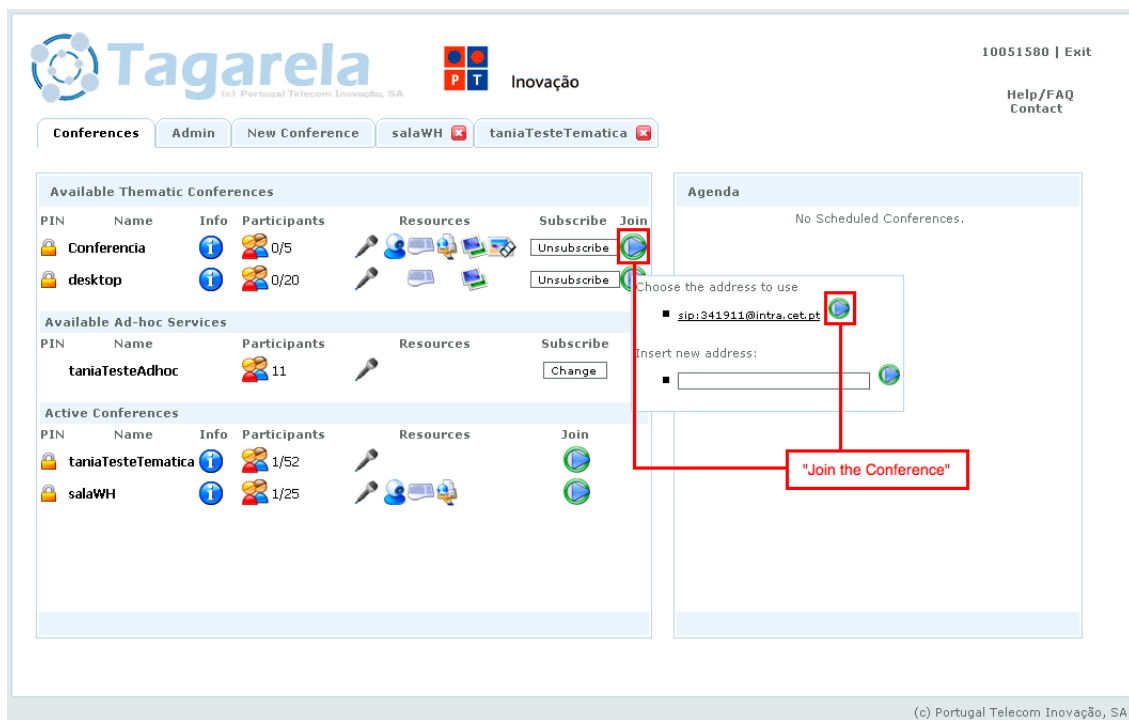


Figure 22: Page with the list of subscribed conferences.

After the user has selected the SIP account, one call will be made to that account, and the call must be answered, in order to join the room. Finally the user joins the room and becomes a participant. At figure 23 is the sequence diagram for joining a conference room, showing the interaction between the *Webhuddle* Server and *Tagarela*.

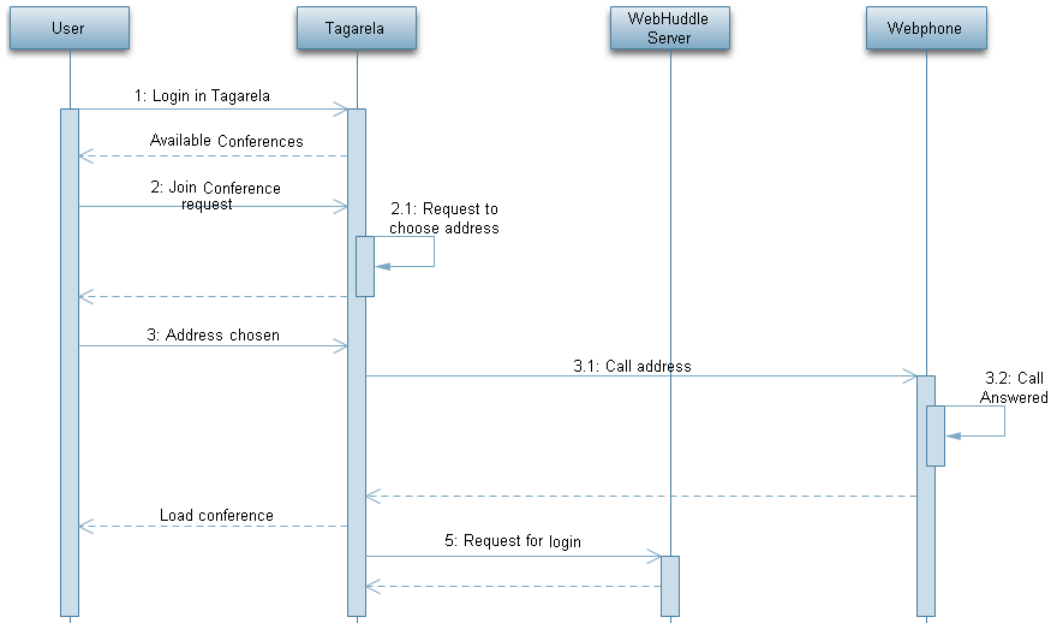


Figure 23: Sequence diagram of joining a conference room, using a Webphone.

Regarding to the SIP account, the user may use several types of phones (physical phone, softphone or webphone), depending on the preferences of the user and the existing information in the database. For example, if the user is using IE and in the database he has an identity associated with him, with the structure `"web:SipUsername@Domain/SipPassword"`, then the user can join the room using the webphone. If, on the other hand, the user is not using IE, although he has that type of entity, he can not use webphone. This happens because the webphone is only available when using IE (this is explained at the section 4.2.3). At last, if the user has only the entity `"sip:SipUsername@Domain"`, he can not use the webphone, whatever browser he is using.

At last, every time that someone joins one conference room the logon URL is sent to the *WebHuddle* Server. This means that a *WebHuddle* session is started to that conference in the participant browser. This is important when the participant wants to start a desktop or a slideshow sharing, because such actions can only be made after the user has done the login at the *WebHuddle* Server.

As it can be seen at figure 23, to join a conference room, first the user needs to login in *Tagarela*, and he/she will get a list of the subscribed conferences (thematic and/or ad hoc). The user must join to the wanted conference room. Next, *Tagarela* will ask the user to choose the SIP address to use.

In this example of sequence diagram (figure 23) is represented the interaction with *Tagarela* when the user is able to use the webphone, instead of another kind of phone. So, when the user choose the conference room to join, *Tagarela* makes a call to the Webphone, that it will be answered automatically. After the call is answered, *Tagarela* will load the conference room page into the browser of the user. Once the user has joined, it is sent a request to *WebHuddle* to login, using the conference name as parameter.

### 4.1.3 Inside of Conference Room

Once an user joined a conference room in *Tagarela*, there is an area where there is a list of the users that are in that room, and the resources that each user has, according to the resources that the room has available (see Appendix D to have a description of the information available at the *Floor Control* area).

The *ModeratorConf* of the room has the ability to remove/add the permissions for participants to use the resources. When a participant wants to use one resource, but does not have permission, needs to request the respective resource, and then the *ModeratorConf* receives the permission request. When receiving the request, the *ModeratorConf* decides if wants to accept the required permission or deny it.

The page of a conference is divided into several areas. There is a *Floor Control* area, where the participants can have access to, for example:

- who else is in the meeting;
- resources that can be used into the conference;
- available resources of each participant;
- chat area;

Another area that exists is the one where the participants can see the video captured from the webcams of the participants that are sharing video. When a video-conference is taking place, is in this section that the video is presented. There is also an area where the participants can see the files that have been shared, in the current conference.

Finally, there is one area where the participants can manage the files to be in the slideshow sharing. In this section, the participants can upload images (.jpeg, .jpg, .gif, .bmp), zip files (with images) and presentations (.ppt, .sxi). The participants can also decide which images they want to present, and can also delete images.

#### 4.1.3.1 Start Desktop Sharing

Figure 24 represents the interactions between participant, *Tagarela* and *WebHuddle* when the participant wants to start a desktop sharing, inside a conference room. The first step is to login in *Tagarela* and choose the conference that the user wants to join. When the page of the conference is being loaded, is made the login of this conference in *WebHuddle*, so that there is a session on that service, to allow the desktop and slideshow sharing, if it will be needed.

During the conference, when a participant pretends to start a desktop sharing, a set of verifications are performed, to see if that request is possible to be attended or not. The function that is used to verify the desktop sharing permission is called *doVerifyDesktopResourcePermission()* and the verifications performed are the following:

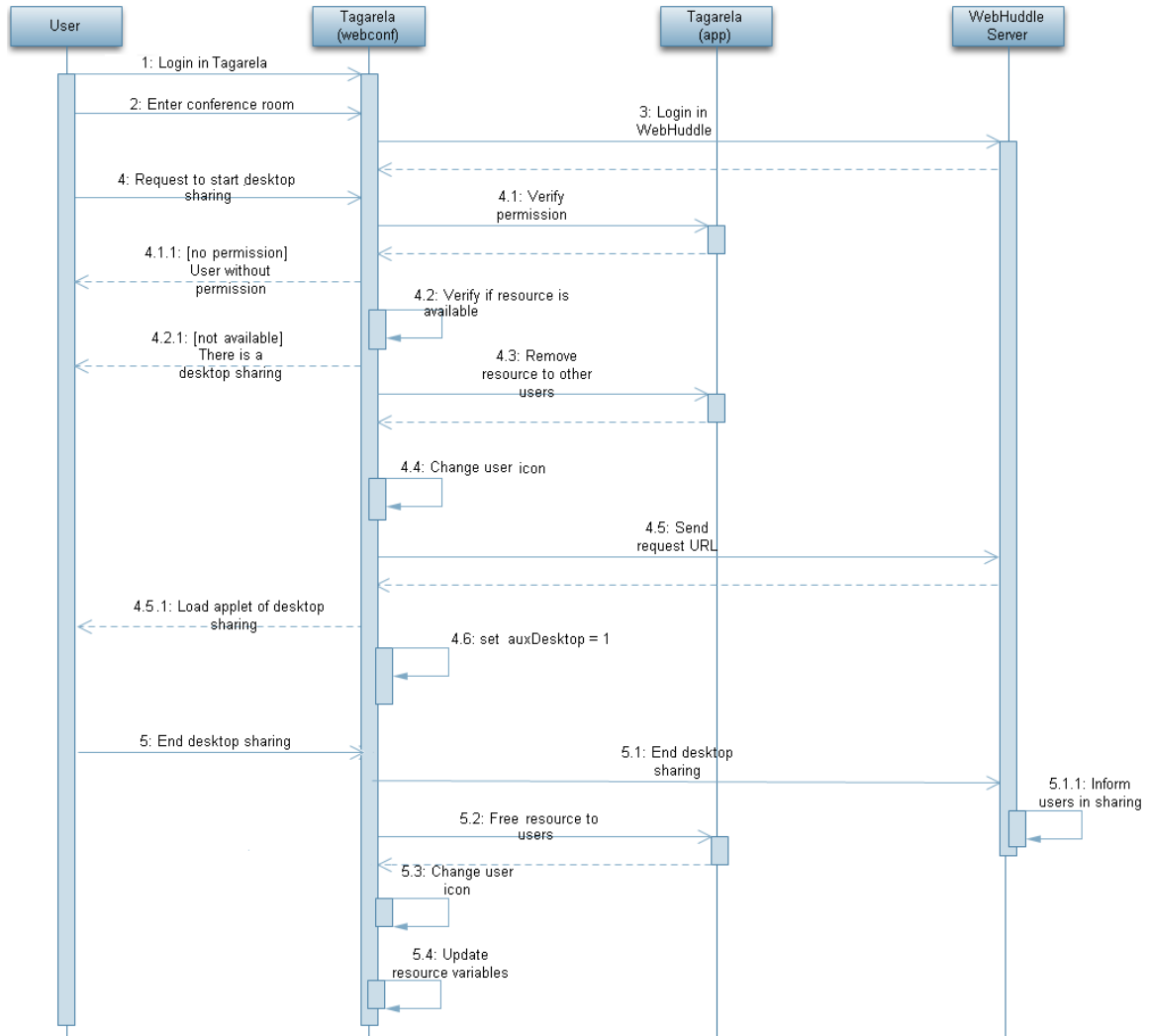


Figure 24: Sequence diagram of starting one desktop sharing, in *Tagarela*.

- participant has permission to use the desktop sharing resource?:
  - if no, then is sent a request to the *ModeratorConf* to request permission;
  - if yes, a *flag* is used to check if there is another desktop sharing ongoing, in that conference room;
    - \* if no:
      - the permission to start desktop sharing is going to be removed from all users, using the function called *doChangeAllPermission()*;
      - the participant icon will have a picture indicating that the participant is sharing the desktop. This action is performed by the function *doSetDesktopSharing()*;
      - using the function *createTabDesktopSharing()* is going to be open a new tab with the

content of loading, through the request URL, the applet responsible for the desktop sharing;

· the *flag* that indicates that there is a sharing ongoing is assigned with a specific value.

This is done to prevent the creation of another desktop sharing, at the conference;

\* if there is already a desktop sharing ongoing, the participant will receive a message to inform about it.

In Appendix E there is more information about the verifications made before start a desktop sharing, and the messages received by a participant, as result of these verifications.

#### 4.1.3.2 Start Slideshow Sharing

Figure 25 represents the interactions between participant, *Tagarela* and *WebHuddle* when the participant wants to start a slideshow sharing, inside a conference room. The first step is login in *Tagarela* and choose the conference that the user wants to join. When the page of the conference is being loaded, is made the login of this conference in *WebHuddle*, so that there is a session on that service, to allow the desktop and slideshow sharing, if it will be needed.

During the conference, when a participant pretends to start a slideshow sharing, a set of verifications are performed, to see if that request is possible to be attended or not. The function that is used to verify the slideshow sharing permission is called *doVerifySlideshowResourcePermission()* and the verifications performed are the following:

- participant has permission to use the slideshow sharing resource?:
  - if no, then is sent a request to the *ModeratorConf* to request permission;
  - if yes, a *flag* is used to check if there is another slideshow sharing ongoing, in that conference room;
    - \* if no, the actual value of the scriptID is going to be get from *WebHuddle*. In the case that scriptID is different from null, its value is going to be used in the request URL of *WebHuddle*. If it is null it means that the conference does not have any script to use in the slideshow sharing, and this parameter is not sent in the request URL.

Whatever the scriptID value, the following steps are made, with the exception that the URL is different, as explained above:

- the permission to start slideshow sharing is going to be removed from all users, using the function *doChangeAllPermission()*;
- the participant icon will have a picture indicating that the participant is sharing a slideshow. This action is performed by the function *doSetSlideshowSharing()* ;



- using the function called *createTabSlideshow()* is going to be open a new tab with the content of loading, through the request URL, the applet responsible for the slideshow sharing;
- the *flag* that indicates that there is a sharing ongoing is assigned with a specific value. This is done to prevent the creation of another slideshow sharing, at the conference;
- \* if there is already a slideshow sharing ongoing, the participant will receive a message to inform about it.

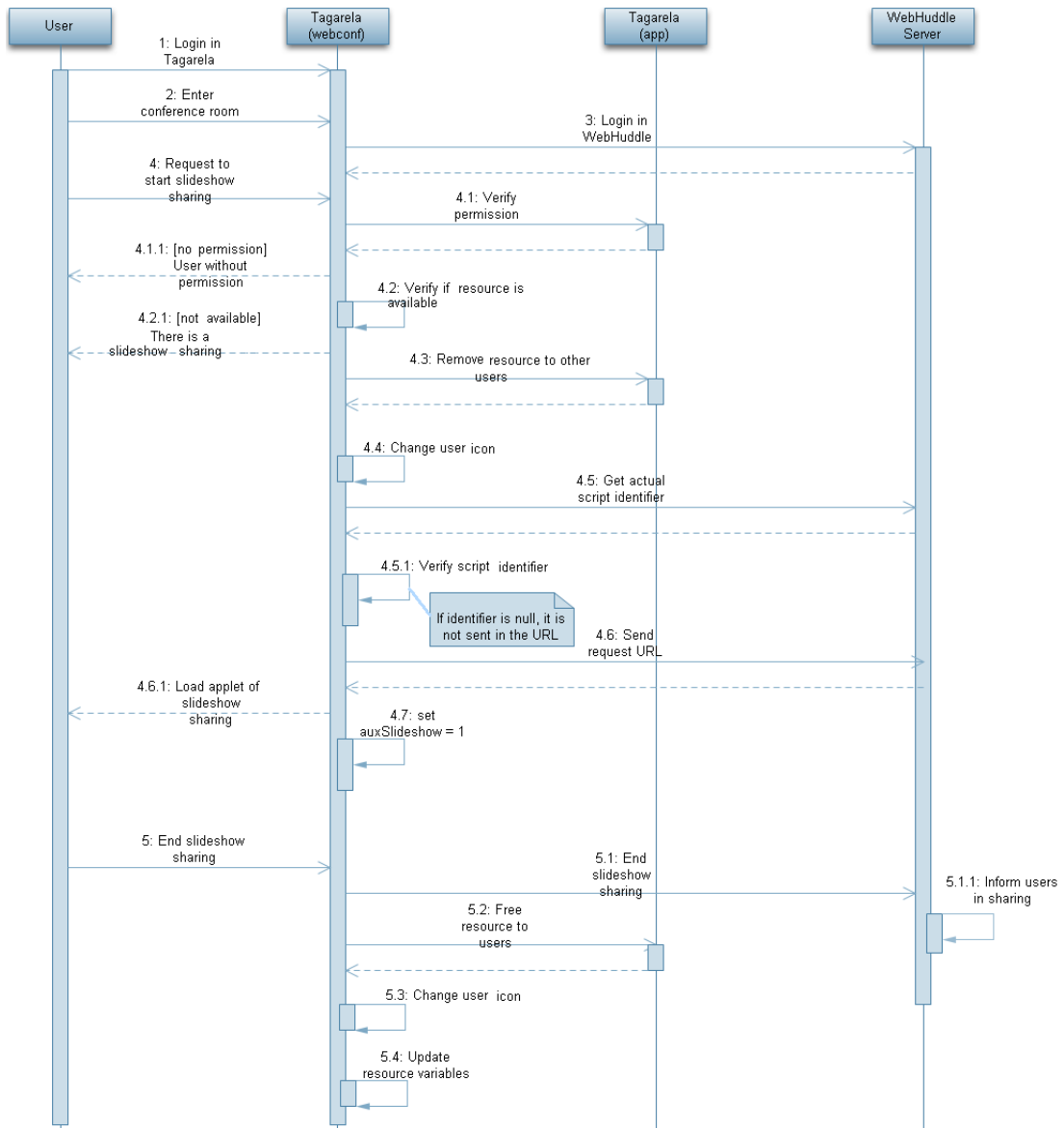


Figure 25: Sequence diagram of starting one slideshow sharing, in *Tagarela*.

In Appendix F there is more information about the verifications made before start a slideshow sharing,

and the messages received by a participant, as result of these verifications.

Because of some difficulties on the implementation, when the *ModeratorWH* wants to change the selected tab (tab of desktop or slideshow sharing), he/she will receive a message saying: "*Because you are moderator of this sharing, leaving this window will end the sharing. Do you really wants to leave?*". If the *ModeratorWH* choose "yes", then the sharing will be ended, and the participant of that sharing will be informed, at the status bar.

This problem happens because when the user changes to another tab, the context of the applet is destroyed. So, when the participant who started the desktop sharing, for example, returns to that sharing tab, the URL responsible for loading the applet is called again, creating a new session of the sharing. And if there are participants who have already joined the sharing, they will be in another session of the sharing, without any *ModeratorWH*. Because of this problem, it was implemented the notification of the *ModeratorWH*, when he/she wants to change of selected tab.

#### 4.1.3.3 Join Desktop or Slideshow Sharing

In order to a participant to be able to join one desktop/slideshow sharing, needs that there is actually one desktop/slideshow sharing going on. See Appendix G for more information about the steps executed by *Tagarela*, when a participant wants to join a desktop/slideshow sharing.

Figure 26 represents the sequence of actions needed to an user to join a sharing (desktop or slideshow). After login, the user needs to choose the conference that he/she wants to join. Once inside the conference, when the user pretends to join a desktop sharing or a slideshow sharing the functions *doJoinDesktop()* and *doJoinSlideshow()* are called, respectively. Inside these functions the following verifications are performed:

- if there is a desktop/slideshow sharing ongoing;
- if the participant is not already in that kind of sharing;

In the case that the sharing is ongoing and the participant is not in it, a request is sent to *WebHuddle*, through an URL. The content resulting from this request will be shown in a new tab, created for this purpose, using the functions *createTabJoinDesktop()* and *createTabJoinSlideshow()*, depending on the case.

When the participant decides to leave the sharing, closing the respective tab, *WebHuddle* detects this action, and removes the user from the sharing. After, *Tagarela* calls *doFreeResource()*, that is responsible for updating the variables associate to that user. The *flags* used to know if the user is in a sharing, will be assigned with a specific value. This will indicate that the user is not in the sharing anymore.

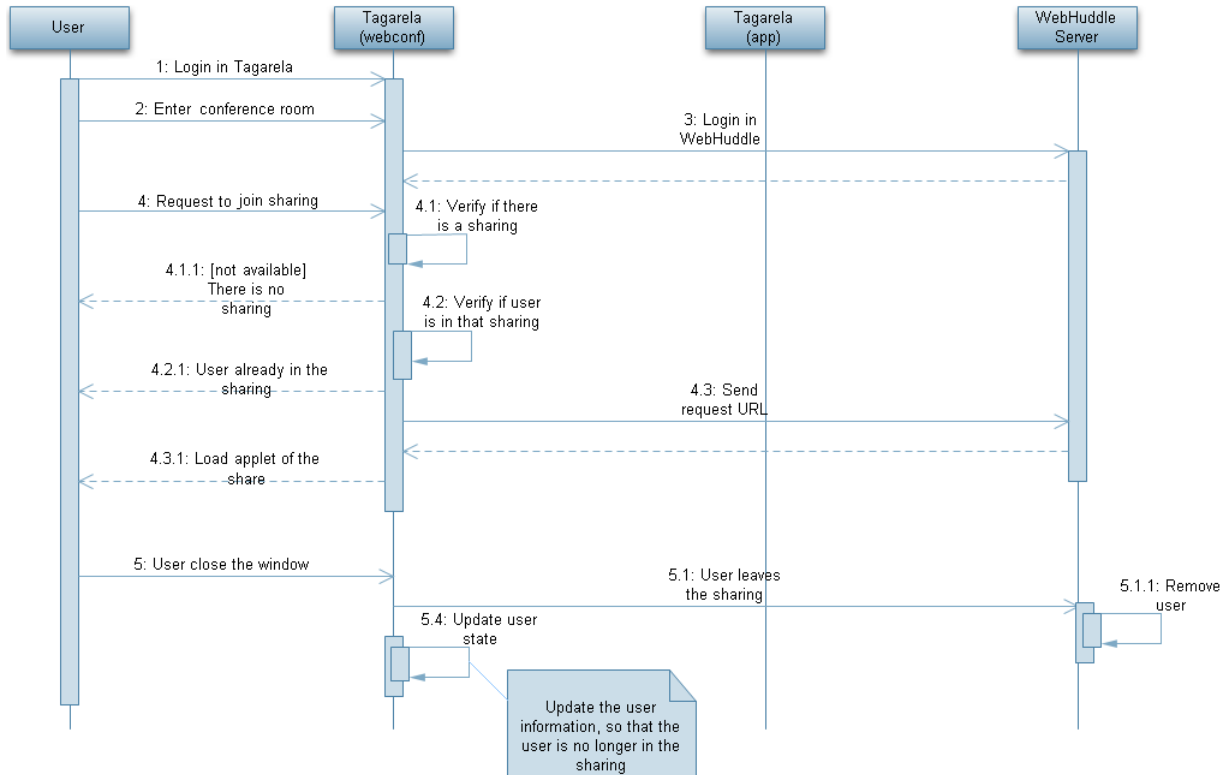


Figure 26: Sequence diagram of joining one desktop/slideshow sharing, in *Tagarela*.

#### 4.1.3.4 Slide Upload and Management for Slideshow Sharing

There is an area, corresponding to the bottom of the page of the conference room, where the participants can upload images to the conference. It is also possible to manage the images that the participant wants to share, in the slideshow sharing. In Appendix H is represented the interface that allow to do this.

As it can be seen in figure 27, before being able to upload files, the user needs to login in *Tagarela* and join a conference room, and the login in *WebHuddle* is consequently made. After these actions, the participant can upload the slides to start a slideshow sharing.

When the user wants to manage the slideshow files associated to the conference room, he/she has access to an interface, that allows the uploading of new slides and managing the existing ones. The interface is obtained after sending the request URL to *WebHuddle* Server, and then the respective applet will be loaded.

Then, when the user wants to upload a new slide, he/she needs to choose the slide and upload it. In the case of management the slides, the participant can decide to delete a slide or change the order of it, or even change the name of the slide.

Whenever any change is made with the slides, the event is sent to *WebHuddle* Server, that updates the script identifier. This script identifier will be used to start a slideshow sharing, when a participant

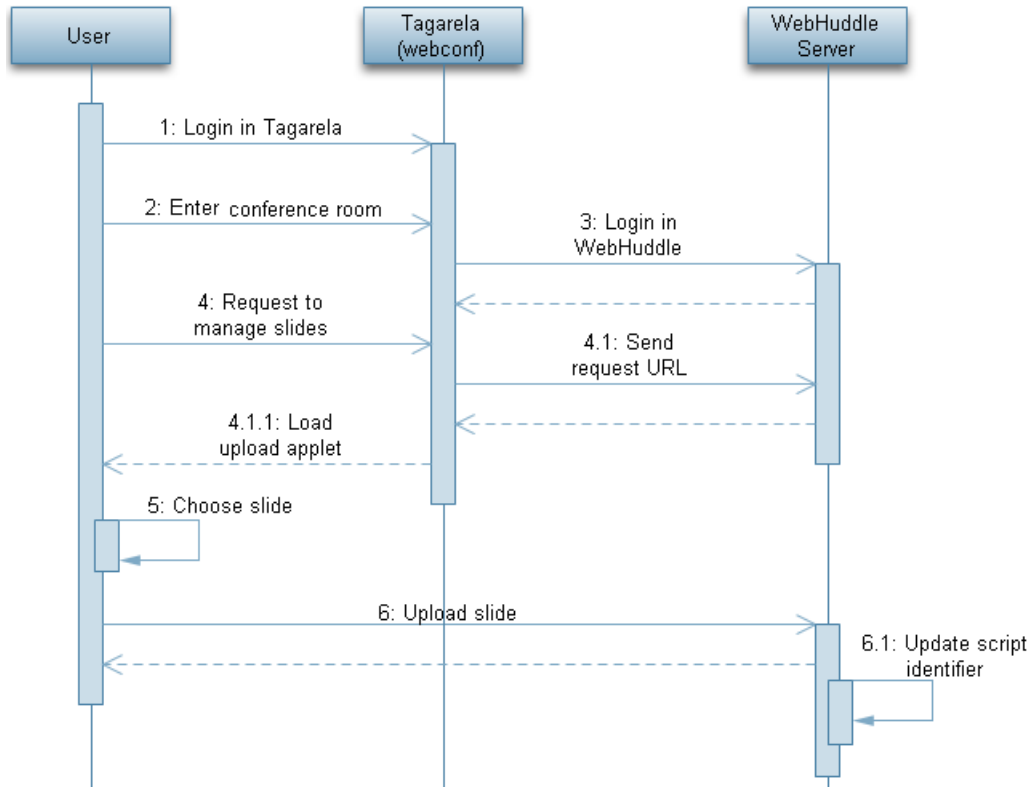


Figure 27: Sequence diagram of uploading images to use in the slideshow sharing, inside one *Tagarela* conference room.

request this action.

#### 4.1.3.5 Request Permission

Once inside a conference room, the *ModeratorConf* can revoke any resource permission to the participants, using the function *doSwitchResourcePermission()*. In the case that the resource revoked is the desktop sharing or the slideshow sharing, and the participant wants to use the resource, he/she can click in one of two buttons: or in the resource next to his/her name, or in the button at the bottom of the *Floor Control* area.

When the participant request the permission to use a revoked resource, the *ModeratorConf* will receive the information shown in figure 28.



Figure 28: *ModeratorConf* receives permission request.

The *ModeratorConf* when sees the request permission may click on the resource and a window will be opened, as the one that is presented in figure 29. Now, the *ModeratorConf* may decide if wants to accept or deny the request permission.

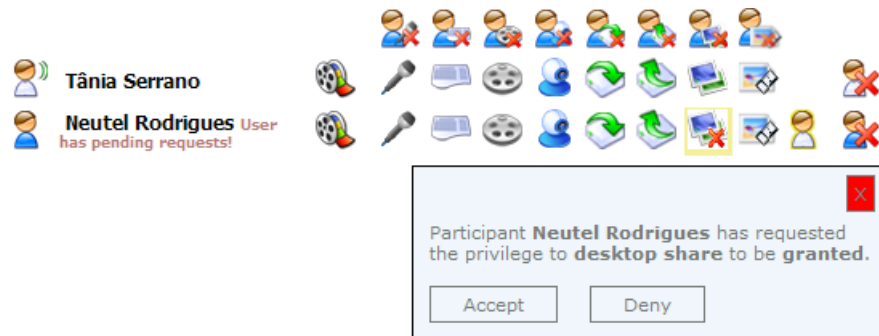


Figure 29: Interface for *ModeratorConf* decide if wants to give permission or not.

## 4.2 Webphone Integration

In order to integrate a webphone into the *Tagarela* webpage it was done an analysis on two webphone Software Development Kit (SDK)s and then it was analysed one *open source* softphone. After implementing some changes on the softphone, it was created one *plug-in* that was then integrated into *Tagarela*.

### Webphone vs Softphone

First of all, it is important to make a distinction between a softphone and a webphone, so that their differences can be understood. Softphone is a software application, that is installed in the computer, and allows to make calls over the Internet, using the mouse or keyboard to dial phone numbers. To use a softphone, the computer must have a sound card, plus a speakers or headset, and a microphone. On the other hand, a webphone has the same feature as a softphone, but with the difference that it does not have to be directly installed in the computer. Instead, the webphone is integrated in the HTML code of a webpage, and automatically downloaded by the browser whenever it is required.

This analysis was made with the aim of replacing an existing webphone solution, at *Tagarela*. The purpose of the webphone, inside the *Tagarela* webpage, is to allow the users to use *Tagarela*, without the need to launch a softphone application. Thus, if the users want to use *Tagarela*, they only need to open the Internet browser.

The two webphones that were tested and analysed were *Abbeyphone Voice Over Web (VOW)* of Abbenet [37] and *eyeP Foundation* of eyeP Media [38]. The softphone which was used to be integrated in *Tagarela* is called *Emansip* [39]. This softphone is based in one *open source* softphone called *Linphone* [40].

### 4.2.1 Analysis of Webphone SDKs

In order to analyse the given webphones, it was provided one list of requirements that the SDKs should meet. Those requirements were (see table 1 with a resume of the analysis):

- comparison with the existing solution;
- possibility of replacement of the used solution at the *Tagarela* service;
- support of the following browsers:
  - IE, version 6 and 7;
  - Firefox;
  - Safari;
  - Opera;
- support in the next OS:
  - Windows XP and Vista;
  - Mac OS;
  - Linux;
- support of earlymedia [41];
- audio/video codecs, including Speex and H.264;
- video resolution: Standard-Definition (SD)/High-Definition (HD);
- installation of the *plug-in* in a way that should be transparent to the user;
- the installation/use should be possible to users without privileges of administration;
- support of SIP Transport Layer Security (TLS) and Secure Real-time Transport Protocol (SRTP).

#### 4.2.1.1 Analysis of Abbeyphone

Considering the set of criteria listed above, only a few are met with *Abbeyphone*. These criteria are:

1. the application works in IE, versions 6 and 7, as in Firefox. However, due to the fact that the application use ActiveX, it only works on Windows XP and Vista, not working in Mac OS or Linux. Because of this limitation, the *Abbeyphone* does not work in Safari and in Opera;
2. the supported codecs are:
  - (a) Audio: G.711;

(b) Video: H.261 and H.263;

3. about the installation of the *plug-in* in a transparent manner, the solution of *Abbeynet*, meets this requirement. The same happens with the installation/use by an user without privileges of administration.

The *Abbeyphone* does not support SIP TLS or SRTP, and there are some criteria for which was not found any information. These criteria are about the support of earlymedia and the support to the video resolution SD/HD.

In the course of testing the application there were some problems. Among them:

- the same code did not worked in all computers that were tested;
- in the computers that worked, when a call was answered, the browser crashed;
- it was only possible to establish a call between the *abbeyphone* and the *eyeBeam*. And this established call only transmitted audio.

#### 4.2.1.2 Analysis of eyeP Foundation

With the webphone called *eyeP Foundation* only the following criteria were verified:

1. the information that is described at the point 1, of the section of Analysis of Abbeyphone, stated above, also applies to this solution. With the exception of that the *eyeP Foundation* does not work with Firefox too;
2. this solution supports the following codecs:
  - (a) audio: G.711 u-Law and A-Law, G.722, G.726, Speex (8 and 16 kHz) and internet Low Bitrate Codec (iLBC);
  - (b) video: H.261, H.263 and H.264.

On the other hand, the requirements which were not met by the *eyeP Media* solution were the following:

- as it happens with *Abbeyphone*, *eyeP Foundation* does not support SIP TLS nor SRTP;
- the installation of the *plug-in* is not done in a transparent manner, like in the *Abbeyphone*. To install of the *eyeP Media plug-in*, it will be needed to download an executable file, which will install the needed libraries. This installation process requires the user to make some decisions, like, for example, select the folder to where the installation should be done. However, the installation/use of this solution can be done by users without administration privileges.

It is worth noting that, the installation of this *plug-in* is done much in the same way that the *plug-in* of *eyeBeam* is done.

Finally, as it happened with *Abbeynet* solution, some information is missing on *eyeP Media* specifications. So, there is no information about:

- support of earlymedia;
- type of video resolution: SD/HD

#### 4.2.1.3 Analysis of Emansip

To end this analysis, the *Emansip* solution met the following criteria:

1. the information that is described at the point 1, of the section of Analysis of Abbeyphone, stated above, also applies to this solution. With the exception of that the *Emansip* does not work with Firefox too;
2. the solution supports the following codecs:
  - (a) audio: G.711, Speex, iLBC, GSM, G.723, G.729, G.726, G.721;
  - (b) video: H.263-1998, H.264, Theora, MPEG4.
3. unlike the other two solutions, *Emansip* supports SIP TLS and SRTP.

On the other hand, the criteria which were not met by the *Emansip* solution were the following:

- as it happens with *Abbeyphone* and *eyeP Foundation*, there is missing information about the type of video resolution;
- is also missing information about the support of earlymedia;
- because *Emansip* is a softphone, instead of being a webphone, there is no *plug-in* to be installed in the computer. So, there is no information about the transparent installation and the installation without administration privileges.

#### 4.2.2 Emansip

*Emansip* is a softphone based on the *open source* softphone, called *Linphone*. With *Emansip* the users can make and receive audio and video calls, as with others softphones. An example of the *Emansip* interface is at figure 30.

The *Emansip* Toolkit is developed in C++ and it uses three main libraries: *amsip*, *eXosip2* and *mediastreamer2*. At table 2 it can be seen some features of each of these libraries.

In order to implement and integrate the webphone, in *Tagarela*, through the *Emansip* softphone, it was provided the *Emansip* toolkit. In the code of this toolkit there is a Graphical User Interface (GUI) to test the *Emansip* features, as register one SIP account, and make and receive calls with video included.



	VOW da Abbeynet	EyeP Media	Emansip
IE 6	✓	✓	N/A
IE 7	✓	✓	N/A
Firefox	✓	✗	✗
Safari	✗	✗	✗
Opera	✗	✗	✗
Windows XP	✓	✓	✓
Windows Vista	✓	✓	✓
Mac OS	✗	✗	✗
Linux	✗	✗	✗
Support to earlymedia (RFC 3960)	N/A	N/A	N/A
Audio / Video Codecs	✓	✓	✓
Video Resolution: SD / HD	N/A	N/A	N/A
Installation of the plug-in in a transparent way	✓	✗	N/A
Installation/use to users without administration privileges	✓	✓	N/A
Support of SIP TLS	✗	✗	✓
Support of SIP SRTP	✗	✗	✓

✓ – Supported.

✗ – Not Supported.

N/A – Not Available.

Table 1: Summary table with the result of the study of the webphones.

Through this application, developed in Visual Basic (VB), it was made some changes to the toolkit, as it is explained in the following section. After this, it was generated a Dynamic Link Library (DLL) that was then used to create an ActiveX object. This object was later used to create a webphone and integrate it into *Tagarela*.

An ActiveX object is a Component Object Model (COM) developed by Microsoft for Windows platforms. Through the use of COM runtime, it is possible to develop software components, that perform functions. Several Microsoft Windows applications, such as IE and Microsoft Office, use ActiveX controls to build their feature set, as well as encapsulate their functionality as ActiveX controls, so that the features can be embedded in other applications. IE also allows the ActiveX controls to be embedded inside web pages.

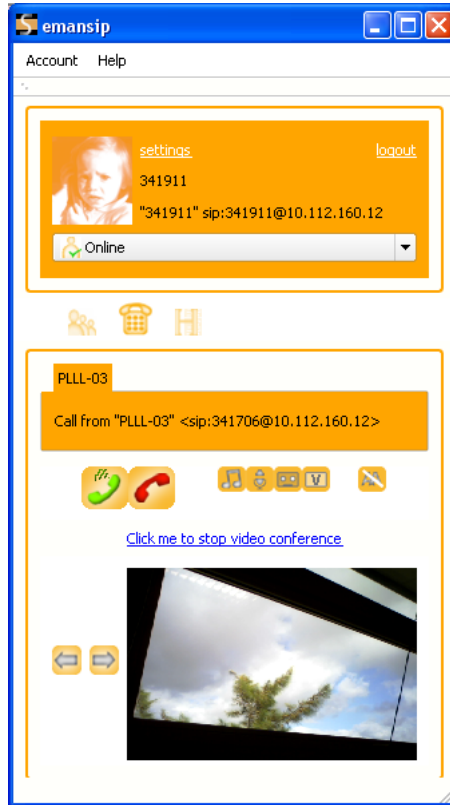


Figure 30: Interface of the *Emansip* softphone.

#### 4.2.3 Webphone Implementation

In the *Tagarela* interface there is an area where it can be seen one icon corresponding to the state of the webphone registration. When using IE, this icon is available and allows that, depending on the color that it has, the user can see if the webphone account is:

- **disabled:** indicates that the webphone is not registered;
- **registering:** informs that the webphone is being registered;
- **ready:** indicates that the webphone is registered and ready to use;

In the *Tagarela* code, when is detected that the user is using IE and he has a webphone account at the database, the application informs the user to wait for the registration of the webphone. Then, it is called a function, called *initWebphone*, with three parameters: *webUser*, *webPassword* and *webDomain*. This values are those that are obtained from the database when the user has an entity started by "web:".

Inside the function *initWebphone()* is created an ActiveX object, with the name "plugin". After, this object is used to call a function, called *initialize()*, with the same parameters that were used for *initWebphone()*. Inside *initialize()* is used the following set of functions:

	amsip	eXosip2	mediastreamer2
<b>Protocol</b>	SIP/SDP, SIMPLE, STUN, RTP	SIP signalling only	-
<b>Security</b>	SRTP, TLS	TLS	-
<b>NAT Transversal</b>	ICE	-	STUN, ICE
<b>Audio Codecs</b>	G.711, Speex, iLBC, GSM, G.723, G.729, G.726, G.721	-	G.711, Speex, iLBC, GSM, G.723, G.729, G.726, G.721
<b>Video Codecs</b>	H.263-1998, H.264, Theora, MPEG4	-	H.263-1998, H.264, Theora, MPEG4
<b>Features</b>	Hold, Mute, DTMF, insert text inside video streams, etc.	Call transfer, call pickup, etc.	Text insertion, recording, resampling, etc.

Table 2: Summary table with the features of the libraries used by *Emansip*.

- **API\_am\_config(webUser, webPassword, webDomain):** function responsible for assigning some global variables, with the arguments received. These values are used during all the execution of the *plug-in*;
- **API\_save\_am\_config("Webphone/amsip\_cfg.xml"):** this function save the values assigned with the previous function, to the file received as parameter;
- **API\_am\_init(pbVideo.Handle):** function that executes a set of steps, that are needed to send and receive video, during a call. The *pbVideo* is the location, inside the HTML page, where the image of the webcam is going to be displayed;
- **API\_am\_option\_find\_out\_sound\_card(sndcard):** obtain the available devices for sound output;
- **API\_am\_option\_find\_in\_sound\_card(sndcard):** obtain the available devices for sound input;

After the execution of the function *initialize()*, from the ActiveX object, it is called a function, from this same object, to register the SIP account, *API\_am\_register()*. This function is only used to call another one, *am\_register\_start()*, at the *amsip* library, of the *Emansip* toolkit. The function *am\_register\_start()* configures the webphone to be registered with the SIP account of the user. When the initialization of the webphone is finished, the user is informed through the next message: "Your webphone is registered and ready to use!". With this, the user may use the webphone to join any conference room.

When the participant wants to join a conference room, using the webphone, only needs to click on the "Join the Conference" button and choose the account of the webphone. After this, the application starts the call to the webphone with the username equals to "webUser", and then the call will be answered. To make possible the integration of the webphone, into *Tagarela*, it was made a change to the way how

the calls were answered by the toolkit. This is, the calls received by the *plug-in* must be answered automatically.

When the event of receiving a call is detected, by the function *API\_process\_event()*, the call is answered automatically. This is done by calling the function *ClickHoldCall()*. The calls that the webphone receives are answered automatically, reducing the number of actions that the user needs to perform, in order to interact with the service.

To end a phone call is used the function *ClickStopCall()*. Inside *Tagarela* this function is called when the participant ends his/her participation on the conference room.

The information in this chapter allowed to understand how the *WebHuddle*'s features were integrated into *Tagarela*, and how does it changed the architecture of *Tagarela*.

The sequence diagrams with the required actions to perform the communication between user, *Tagarela* and *WebHuddle*, were presented and explained.

At the end of this chapter, the webphone integration was presented. It was explained how the development of the softphone in a webphone *plug-in* was made.

## 5 Tests

To assess the effectiveness of adding the three new features into *Tagarela*, several functional tests were performed. The list of those tests is presented in the following section (section 5.1). These tests only check the functionality of the product, since this Thesis represents a proof of concept of adding three new features to a web conferencing service: desktop sharing, slideshow sharing and webphone.

To the realization of the tests two computers were used: one that was the server and other that was the client. The required tools were an Internet browser, Mozilla Firefox or IE, depending on the type of test that was being done, and a softphone or a webphone with a SIP account registered, also depending on the test.

After listing the functional tests, is presented the required procedure to realize the tests successfully (see section 5.2). Some of the tests require that another test is done first. But these situations are explained during the test procedure. Finally, in section 5.3 it can be seen the result of tests that were made, and the conclusions that can be drawn from these results.

### 5.1 List of Functional Tests

The list of the functional tests is presented bellow. These tests allows to verify if all the features added to *Tagarela* are working or not, doing a set of actions.

1. T1. Create new thematic conference, with desktop and/or slideshow sharing resources;
2. T2. Create new ad hoc conference, with desktop and/or slideshow sharing resources;
3. T3. Register account of webphone, after using IE to login in the *Tagarela* homepage;
4. T4. Use the webphone to join one conference room;
5. T5. Use a softphone to join one conference room;
6. T6. Start one desktop sharing;
7. T7. Start one slideshow sharing;
8. T8. Join one desktop sharing;
9. T9. Join one slideshow sharing;
10. T10. Ask permission to use the desktop sharing resource;
11. T11. Ask permission to use the slideshow sharing resource;
12. T12. Upload one presentation;

13. T13. Try to start a desktop/slideshow sharing, when there is one already underway;
14. T14. Try to join one desktop/slideshow sharing, when there is no sharing going on;
15. T15. Manage the folder of slides to use in a slideshow sharing (add or remove slides);
16. T16. Send/receive audio through the webphone, inside a conference;

## 5.2 Test Procedure

The realization of all the following test cases presupposes that the user is already authenticated in *Tagarela*. To do this, the user need to login in *Tagarela*, writing the *login* and *password* values on the webpage, choosing the language to use (Portuguese or English) and then clicking on the "Entrar" button.

Although the login process is not described in the test procedures, the login is a necessary part of all the listed tests. Hence, and to avoid unnecessary repetition, the login was described above. However, the number of actions needed to login are part of the number of actions needed to execute the tests, as well as in the time of execution, that are described in the test results (see section 5.3).

### **T1. Create new thematic conference, with desktop and/or slideshow sharing resources**

In order to perform the action of creation a new thematic conference room, with desktop and/or slideshow sharing resources, the following actions are needed:

1. On the main page, select the "Admin" tab and choose the Enterprise in which the conference room will be created;
2. Click on "Create New Conference" button at the "Available Thematic Conferences" area;
3. Fill all the parameters and check the desktop and/or slideshow resource, and click on the "Create Conference" button.

### **T2. Create new ad hoc conference, with desktop and/or slideshow sharing resources**

In order to perform the action of creation a new ad hoc conference room, with desktop and/or slideshow sharing resources, the following actions are needed:

1. On the main page, select the "Admin" tab and choose the Enterprise in which the conference room will be created;
2. In the "Available Ad-hoc Services" area, click on "Create New Service" button;
3. Fill all the parameters and check the desktop and/or slideshow resource, and click on the "Create Conference" button.

### **T3. Register account of webphone, after using IE to login in the *Tagarela* homepage**

If the *Tagarela* user is using the IE browser, and has a webphone account available at the database, he/she can use a webphone to interact with *Tagarela*. Next the actions needed to register the user webphone will be explained.

1. After login, it will appear the following message: "*Wait for the registration of your webphone, please!*". When the "OK" button is clicked, the registration of the webphone will be performed;
2. While the registration is being done, an icon at the top of the page will be orange, to indicate that the webphone registration is going on (see section 4.2.3 for more information);
3. When the registration is successful, the next message will appear: "*Your webphone is registered and ready to use!*". Is needed to click on the "OK" button to continue using the *Tagarela*;
4. The button that indicates the state of the webphone, at the top of the page, will turn green. This means that the webphone is registered and ready to use (see section 4.2.3 for more information).

### **T4. Use the webphone to join one conference room**

To be able to use the webphone to join one conference room, the webphone need to be previously registered. The process of the webphone registration was explained in T3. After the webphone registration, in order to join one conference room, the following actions are needed:

1. On the main page, choose the conference that wants to join, by clicking on the respective "Join the Conference" button;
2. A *pop-up* window will appear, where must choose what address wants to use. Is presented an address that corresponds to the webphone account, and is also possible to write a new address. But, because this test is about joining a conference with the webphone, click on the "Join the Conference" button of the webphone account;
3. After choosen the webphone address, the service makes a call to that number, that will answer automatically;
4. When the service receives the event that the call have been answered, the conference page is loaded, and the user joins the conference room, becoming a participant.

### **T5. Use a softphone to join one conference room**

To realize the test of joining a conference room using a softphone with a SIP account, the following actions must be performed:

1. Start a softphone with a registered SIP account;
2. On the main page, choose the conference in which wants to join, by clicking on the respective "Join the Conference" button;
3. A *pop-up* window will appear, where must choose what address wants to use. Is presented an address that corresponds to the webphone account, and is also possible to write a new address. Because this test is about joining a conference with a softphone, must write a new address, for example *341916* (value of a registered SIP account), and click on the respective "Join the Conference" button;
4. When the softphone receives the event call, must answer the call.

#### **T6. Start one desktop sharing**

In order to be able to start a desktop sharing, the following actions need to be performed:

1. Join one conference room, which must have the desktop sharing resource available (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone);
2. Once inside the conference room, click on the "Start Desktop Sharing" button, and a new tab will open with the content of the desktop sharing applet.

#### **T7. Start one slideshow sharing**

To be able to start a slideshow sharing, the following actions need to be performed:

1. Join one conference room, which must have the slideshow sharing resource available (see tests T4. or T5., depending if wants to enter in the conference using a webphone or a softphone);
2. Once inside the conference room, go to the "Slideshow Resource" area, and select the *script* which wants to use by clicking on the "Select Script" button, choosing the *script* and then click on the "OK" button;
3. Click on the "Start Slideshow Sharing" button, at the *Floor Control* area, and a new tab will open with the content of the slideshow sharing applet.

#### **T8. Join one desktop sharing**

In order to be able to join a desktop sharing, the following actions need to be performed:

1. Enter in one conference room, which must have the desktop sharing resource available (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone);
2. Once inside the conference room, click on the "Join Desktop Sharing" button, and a new tab will open with the content of the desktop sharing applet.



### **T9. Join one slideshow sharing**

To be able to join a slideshow sharing, the following actions need to be performed:

1. Join one conference room, which must have the slideshow sharing resource available (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone);
2. Once inside the conference room, click on the "Join Slideshow Sharing" button, and a new tab will open with the content of the slideshow sharing applet.

### **T10. Ask permission to use the desktop sharing resource**

In order to be able to start a desktop sharing when the user does not have permission, the following actions need to be performed:

1. Join one conference room, which must have the desktop sharing resource available (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone);
2. Once inside the conference room, wait for the *ModeratorConf* to revoke the desktop permission;
3. Click on the "Start Desktop Sharing" button, and because the user does not have permission to start a desktop sharing, a permission request is sent to the *ModeratorConf*;
4. *ModeratorConf* receives the request and decides if he/she wants to give permission or not;
5. After the *ModeratorConf* accepted or denied the request permission, the resource icon will reflect the choice of *ModeratorConf*.

### **T11. Ask permission to use the slideshow sharing resource**

To be able to start a slideshow sharing when the user does not have permission, the following actions need to be performed:

1. Join one conference room, which must have the slideshow sharing resource available (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone);
2. Once inside the conference room, wait for the *ModeratorConf* to revoke the slideshow permission;
3. Click on the "Start Slideshow Sharing" button, and because the user does not have permission to start a slideshow sharing, a permission request is sent to the *ModeratorConf*;
4. *ModeratorConf* receives the request and decides if he/she wants to give permission or not;
5. After the *ModeratorConf* accepted or denied the request permission, the resource icon will reflect the choice of *ModeratorConf*.

### **T12. Upload one presentation**

In order to upload one presentation for later use in a slideshow sharing, the following actions need to be executed:

1. Join one conference room, which must have the slideshow sharing resource available (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone);
2. Once inside a conference room, in the "Slideshow Resources" area, is an interface that allows to upload new files;
3. To upload the presentation, browse the computer to select the desired presentation;
4. After choose the presentation, must click on the "Upload" button.

### **T13. Try to start a desktop/slideshow sharing, when there is one already underway**

To realize this test case, the following actions need to be performed:

1. Join one conference room, which must have the desktop/slideshow sharing resource available (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone);
2. Once inside the conference, click on the "Start Desktop Sharing"/"Start Slideshow Sharing" button;
3. Because there is already a desktop/slideshow sharing going on, it will appear the next message: *"There is a desktop/slideshow sharing"*.

### **T14. Try to join one desktop/slideshow sharing, when there is no sharing going on**

To realize this test case, the following actions need to be performed:

1. Join one conference room, which must have the desktop/slideshow sharing resource available (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone)
2. Once inside the conference, click on the "Join Desktop Sharing"/"Join Slideshow Sharing" button;
3. Because the user is already in a desktop/slideshow sharing, it will appear the next message: *"User is already in this desktop/slideshow sharing"*.

### **T15. Manage the folder of slides to use in a slideshow sharing (add or remove slides)**

In order to manage the folder of slides to use in a slideshow sharing, the following actions must be performed:

1. Join one conference room, which must have the slideshow sharing resource available (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone);

2. Once inside the conference, at the "Slideshow Resources" area is an interface that allows to manage the slides available;
3. In this interface click on the "Select Script" button and choose the *script* that wants to manage;
4. After clicking on the "OK" button, choose the *slide* to delete and click on the "Delete" button;
5. Finally, click on the "Save" button to update the information in the database.

#### **T16. Send/receive audio through the webphone, inside a conference**

To send and receive audio through the webphone, once inside a conference room, the following actions must be performed:

1. Proceed with the webphone registration (see test T3.);
2. Join one conference room (see tests T4. or T5., depending if wants to join the conference using a webphone or a softphone);
3. Once inside the conference room, if the participant has the permission to use the audio resource, it is possible to send and receive audio through the webphone.

### **5.3 Results and Analysis**

Before presenting the test results, there are some details that need to be mentioned. Almost all the tests were made using two computers. One computer was the server, because is where the service was running, and the other computer represents the client of the service. The only exceptions were the tests T3, T4 and T15. Because these tests were the ones that included the webphone, and the *plug-in* was only installed on the server.

Another important information is about tests T10 and T11, because the *ModeratorConf* only can revoke a resource permission after the participant have joined the conference room. So, to do these tests, the participant need to join the conference, and wait that the *ModeratorConf* has removed the permission.

#### **5.3.1 Number of Required Actions**

In this section are presented the results of the usability tests. In figure 31 it can be seen the number of actions that are required, in order to perform the respective tests. Concerning to tests T1 and T2, although in those tests must write various parameters, this writing was considered only as one action.

Through figure 31 it can be seen that the tests T7, T12 and T15 are those that need more steps to be executed. This is no coincidence, because these tests are testing the functionalities that are associated with slideshow sharing. So, it is also need to choose the slides that wants to share, besides clicking on

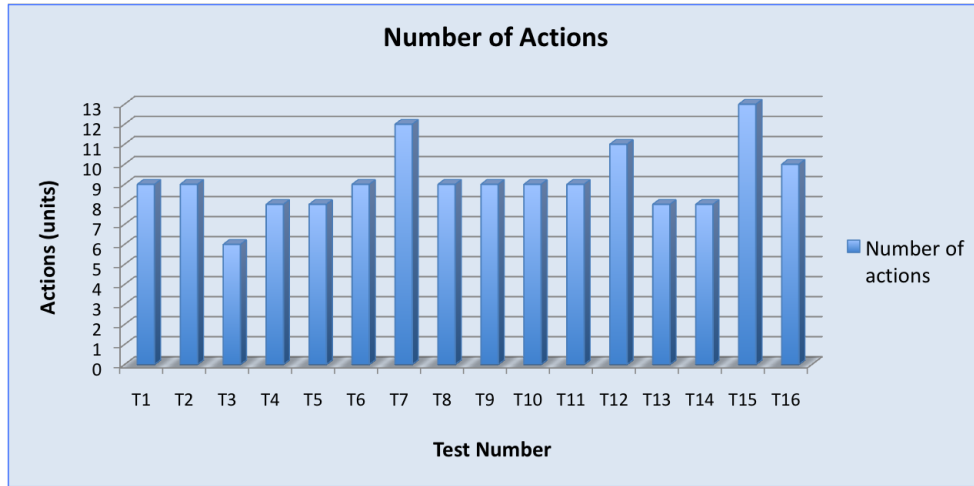


Figure 31: Chart with the representation of the tests of usability result.

the same buttons that are used in desktop sharing. Apart from these three exceptions, the most tests only need nine steps to be executed.

Concluding, through figure 31 it may be seen that, concerning to the usability, the adding of the new features in *Tagarela* was successfully, because the new features can be used as easily as the original ones.

### 5.3.2 Duration of the Tests

Another type of tests that was done had as purpose the measurement of time. For each test five measurements were made, with the aim of having a more reliable result. In figure 32 is depicted the maximum, minimum and average values of the measurements made for each test.

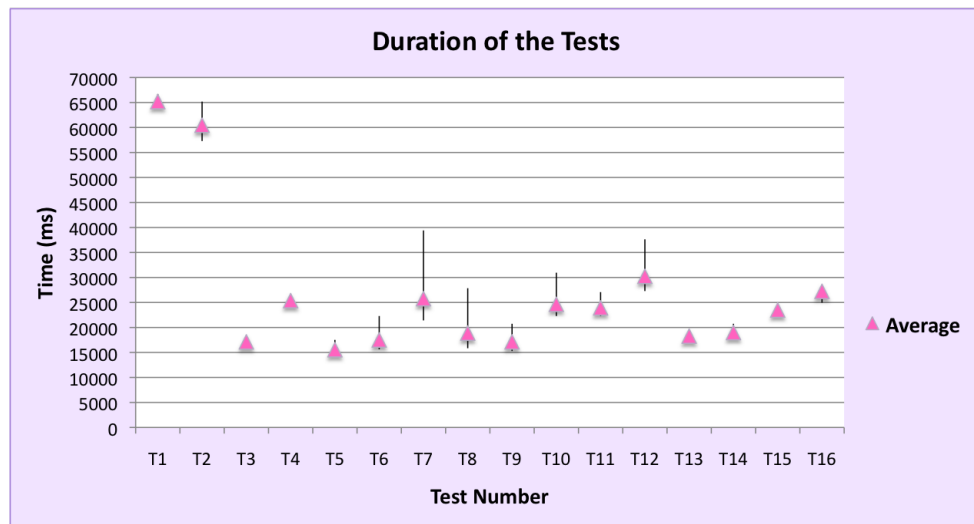


Figure 32: Chart representing the maximum, minimum and average duration of each test.

Through this figure it can be seen that T1 and T2 are the tests that need more time to be executed. This is due to the fact that the user must fill several parameters to create the conference room. However, T1 and T2 have a difference of a few seconds, because T2 has less parameters to fill, and so it takes less time.

There is also a considerable difference in the times of T6 and T7, because for T7 the service need to load the images to the applet, while to start the desktop sharing is only need to load the applet. And if the number of slides is big, more time will be needed to start the slideshow sharing.

Another conclusion that can be taken from figure 32 is that the times of test T4 and T16 are similar. This happens because the only difference between them is the fact that the user need to say something, to be heard at the conference. So test T16 is supposed to take longer than T4, but only a few seconds.

About tests T10 and T11, they also need more time to be executed because, once inside the conference room, the participant have to wait for the *ModeratorConf* to remove the permission, and then ask for it.

Another conclusion that can be taken from figure 32 is that the execution time for test T12 depends on the number of pages that the uploaded presentation has. So, the greater the number of pages, the longer it will take for the presentation uploading. Finally, the time that the test T15 needs to be done, with successful, depends on the number of slides that the participants wants to modify. For this test, it was selected the folder that wanted to share, and then it was deleted only one slide.



## 6 Conclusions

This dissertation proposes to investigate and test the use of the Web 2.0 concepts (e.g., creation and generation of service and contents by users themselves) on the platforms for services to the networks ALL-IP 3GPP IMS.

During this document it was explained the technologies that were studied and used, how was implemented the integration of two specific services, and finally the list of the executed tests to verify the functionality of the new features were presented.

Through the tests it was verified that the new features are easy to use. They do not only need relatively few actions to be executed, nor are slow to be made. The majority of the tests need around nine actions, including the login where the user need to write the username, password and desired language.

With this dissertation the *Tagarela* service was enriched with three new features: desktop sharing, slideshow sharing and a webphone. These features are important, because it is unusual for a conference service to have features that are not only audio, video and chat conversation.

### 6.1 Future Work

Before the sharing service is started, the service must perform some verifications. These verifications are required to know if the participant has permission to use the resource, or if the resource is not being used already. Depending on the result of the verifications, the participant may be informed that can not use the resource, or that the sharing is already started.

A possible feature that could be studied and implemented as future work is replacing the way how the transmission of the content of the new resources is made. Instead of using HTTP to verify resource permissions and transmit the content of the sharings, it could be used SIP to do the verifications and the content of the sharing could be sent inside video stream.

After the integration of the video feature in the webphone *plug-in*, the information exchanged during the desktop sharing and the slideshow sharing can be sent as video stream. In order to do this, instead of doing the resources admission via HTTP, this control could be done using SIP. A possibility of implementation could be through the following steps:

- when a participant wants to start a desktop or slideshow sharing, could be sent an INVITE message to all the participants that are in the conference room;
- in this INVITE message it will be sent, within the video header, the information about the sharing;
- when the participants receive the INVITE message, may decide if want to send the OK message, accepting the invite to join the sharing;
- when the *ModeratorWH* decides to end the sharing, the respective BYE message will be sent to the participants inside that sharing.

Besides this, it is also needed to end the development of the webphone *plug-in*. Due to the lack of time the *plug-in* does not have the video feature, only allowing audio conversation. One possible solution to implement this feature is, after capturing the video from the webcam, present it in the *Tagarela* webpage. Once inside the conference room, the communication with the webphone is established, transmitting audio and video to the room.

There is still another feature that could be implemented in future work. If this service is used, for example, in a Call Center, whenever a customer calls with a question all the interaction can be recorded, including the desktop sharing. This recording can be later sent to the customer, and if the customer ever again has the same problem, he/she will not need to call to the Call Center. The implementation of this feature could be done using the *WebHuddle*, since it is already available.



## References

- [1] <http://www.hi5.com/> (on-line September 2008)
- [2] <http://www.youtube.com/> (on-line September 2008)
- [3] <https://www.blogger.com/> (on-line September 2008)
- [4] <http://www.google.com/ig> (on-line September 2008)
- [5] O'Reilly, Tim, " *What Is Web 2.0*", September 30, 2005.
- [6] Johnston, Alan B., "textitSIP: understanding the Session Initiation Protocol", 2nd edition, 2004.
- [7] IETF RFC 2543: " *SIP: Session Initiation Protocol*", March 1999.
- [8] IETF RFC 3261: " *SIP: Session Initiation Protocol*", June 2002.
- [9] IETF RFC 2616: " *Hypertext Transfer Protocol – HTTP/1.1*", June 1999.
- [10] IETF RFC 821: " *Simple Mail Transfer Protocol*", August 1982.
- [11] IETF RFC 1738: " *Uniform Resource Locators (URL)*", December 1994.
- [12] IETF RFC 2396: " *Uniform Resource Identifiers (URI): Generic Syntax*", August 1998.
- [13] IBM RedBooks, " *Developing SIP and IP Multimedia Subsystem (IMS) Applications*", 1st edition, February 2007.
- [14] IETF RFC 4566: " *SDP: Session Description Protocol*", July 2006.
- [15] IETF RFC 1889: " *RTP: A Transport Protocol for Real-Time Applications*", January 1996.
- [16] Radvision, " *Session Initiation Protocol (SIP), Technical Overview*", April 2005.
- [17] Lauretti, Samuel R., " *Evolução das Redes de Telecomunicação: Arquitetura IMS*", December 06, 2004.
- [18] 3GPP TS 23.228: " *IP Multimedia Subsystem (IMS)*".
- [19] ITU-T H.248: " *Gateway control protocol*".
- [20] PT Inovação, " *SHipNET*", <http://www.ptinovacao.pt> (on-line July 2008)
- [21] PT Inovação, " *ip-Jib Whitepaper*". (on-line January, 2008)
- [22] " *JAIN SLEE Tutorial - Introduction JAIN SLEE*", <http://jainslee.org/downloads/jainslee-tutorial-04.pdf> (on-line August)

- [23] IETF RFC 5239: "A Framework for Centralized Conferencing", June 2008.
- [24] IETF RFC 3265: "Session Initiation Protocol (SIP)-Specific Event Notification", June 2002.
- [25] "Java BluePrints: Model-View-Controller", <http://java.sun.com/blueprints/patterns/MVC-detailed.html> (on-line August, 2008)
- [26] PT Inovação, "Conferência e Colaboração - Descrição Técnica", Version 1.0.
- [27] "3GPP TS 24.147, Conferencing using the IP Multimedia (IM) Core Network (CN) subsystem", Stage 3, Release 7.
- [28] IETF RFC 4353: "A Framework for Conferencing with the Session Initiation Protocol (SIP)", February 2006.
- [29] IETF RFC 4579: "Session Initiation Protocol Call Control - Conferencing for User Agents", August 2006.
- [30] IETF RFC 4575: "A Session Initiation Protocol (SIP) Event Package for Conference State", August 2006.
- [31] IETF RFC 4597: "Conferencing Scenarios", July 2006.
- [32] IETF RFC 2445, *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*, November 1998.
- [33] <http://www.webhuddle.com>. (on-line November, 2007)
- [34] <http://www.jboss.org>. (on-line July, 2008)
- [35] [http://www.oracle.com/technology/sample\\_code/tech/java/j2ee/jintdemo/tutorials/Struts.html](http://www.oracle.com/technology/sample_code/tech/java/j2ee/jintdemo/tutorials/Struts.html) (on-line August, 2008)
- [36] IETF RFC 1631: "The IP Network Address Translator (NAT)", May 1994.
- [37] <http://www.abbeyphone.com/>.(on-line May, 2008)
- [38] <http://www.eyepmedia.com/products/toolkits/>. (on-line May, 2008)
- [39] <http://www.antisip.com/>. (on-line June, 2008)
- [40] <http://www.linphone.org/index.php/eng>. (on-line June 2008)
- [41] IETF RFC 3960, *Early Media and Ringing Tone Generation in the Session Initiation Protocol (SIP)*, December 2004.

# Appendices

## A. More information about SIP

As was already told, SIP messages can be of two kinds: Request and Response. These messages have a common format which is represented in figure 33. The messages are composed of three parts: start-line, message-header and message-body.

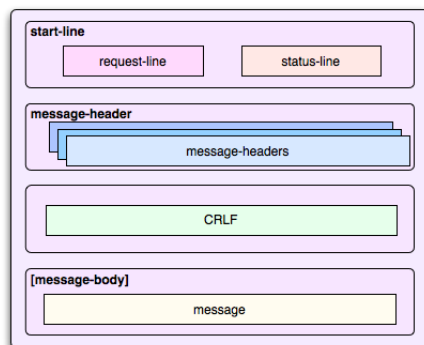


Figure 33: SIP message structure.

In the SIP messages the start-line can be a request or just a status line. The request-line is used to define the type of the request. On the other hand, the status-line indicates if the processing of the request is successful or not.

The headers of SIP messages are made of fields with groups of name and value. There are some fields that are optional, like *content-type* and *length*, but there are obviously some mandatory fields for all SIP messages, like *To*, *From* and *Call-ID*. The table 3 represents this mandatory fields.

Finally, the message body describes the session that is going to be initiated. For example, in a multimedia session the *body* may include audio and video codec types. Message bodies can exist in request and/or in response message. An advantage of SIP is that it makes a distinction between information used for signaling, inserted in the SIP *start-line* and *headers*, and session description information.

### Requests

The SIP request messages, as all SIP messages, follows the structure of a SIP message represented on figure 33, but the fields have different values depending the type of message. At the *start-line* the message have a *Request-Line*, whose format is on figure 34. It is formed by three fields separated by a single space character.

- **Method:** In the case of a SIP request message, this field can be equal to REGISTER, INVITE, ACK, CANCEL, BYE or OPTIONS depending the method to be performed. These are not the

Field name	Description
<b>To</b>	Address of SIP request destination.
<b>From</b>	Corresponds to the origin of the request.
<b>CSeq</b>	Command sequence that make sure that the order is maintained.
<b>Call-ID</b>	Randomly generated string that uniquely identify SIP sessions. This field is used by proxy servers to identify the session to which it belongs.
<b>Via</b>	Has information about where the message should pass, when it moves between caller and callee.
<b>Contact</b>	Has the actual location of the callee. This information can be different from the address in the <i>From</i> header.

Table 3: Mandatory headers.

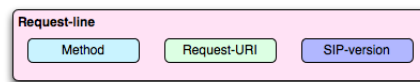


Figure 34: Request-line of the start-line.

only methods that are defined. SIP extensions, that are documented in other RFCs, have others additional methods defined. The table 4 have a simple description of the methods previously mentioned;

- **Request-URI:** Field corresponding to a SIP URI or multiples SIP URIs. It indicates the user/service for which the request is addressed;
- **SIP-version:** Identifies the version of SIP protocol in use.

Method	Description
<b>REGISTER</b>	Gives Registrar the information about the UA's location for incoming SIP requests. When the UA change is location, it must be sent another REGISTER message to performe the update of the Registrar database.
<b>INVITE</b>	Initiate a communication session between two UA's. This message can also be used to initiate a multi party call.
<b>ACK</b>	Used for acknowledgement, indicating that a final response has been received.
<b>CANCEL</b>	Terminate pending requests. A INVITE message can be canceled while a final response isn't received.
<b>BYE</b>	Method that terminates a session.
<b>OPTIONS</b>	Method used to query a server about is capabilities, for example, if it can support a type of media.

Table 4: Examples of request methods.

## Responses

When a UA or a proxy server receives a request message, it originates a response that is sent as a SIP response message. This type of message is different from the request ones, because they have status-line in their start-line, instead of request-line. The status-line have also three fields (see figure 35).

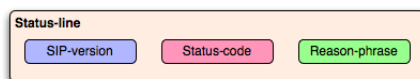


Figure 35: Status-line of the start-line.

- **SIP-version:** Identifies the version of the SIP protocol in use;
- **Status-code:** is a three digit code that represents the result of request processing. This code can be a number between 100 and 699 and all response messages have one of this numeric code associated. The codes have two types of responses and six classes, and are partly based on the HTTP response codes. This types and classes of codes are represented on table 5 and table 6;
- **Reason-phrase:** is a textual description of the *status-code*. While the *status-code* is to be used on the machine processing, the *reason-phrase* is a message to be readable by humans.

Types	Description
Provisional (1xx class)	Responses of this type are used by the servers to indicate some progress, but this do not terminate SIP transaction.
Final (2xx, 3xx, 4xx, 5xx and 6xx classes)	Responses of this type terminate SIP transaction.

Table 5: Types of response messages.

Classes	Description
1xx - Provisional	The request was received; continue to process the request.
2xx - Success	The action was correctly received and accepted.
3xx - Redirection	The next actions that are needed to complete the request.
4xx - Client Error	The received request have bad syntax or the server cannot fulfill this request.
5xx - Server Error	The server did not fulfill a valid request.
6xx - Global Failure	Any server fulfilled the request.

Table 6: Classes of response messages.

## B. Example of tables from WebHuddle Server database

- **CREATE MEMORY TABLE CUSTOMERS** (**CUSTOMER\_ID** INTEGER NOT NULL, **FIRST\_NAME** VARCHAR(256),  
**ORIGINAL\_IP** VARCHAR(256), **EXTERNAL\_ID** VARCHAR(256), **EMAIL** VARCHAR(256), **CREATION** TIMESTAMP,  
**HAS\_ACCOUNT** BOOLEAN NOT NULL, **PASSWORDHASH** VARBINARY, **SALT** VARBINARY,  
**LOCALE\_LANGUAGE** VARCHAR(256), **LOCALE\_COUNTRY** VARCHAR(256), **COUNTRY** VARCHAR(256),  
**CONSTRAINT** PK\_CUSTOMERS PRIMARY KEY(CUSTOMER\_ID));
- **CREATE MEMORY TABLE LOGONS** (**LOGON\_ID** INTEGER NOT NULL, **LOGON\_IP** VARCHAR(256),  
**USERAGENT** VARCHAR(256), **DATE** TIMESTAMP, **CUSTOMER\_ID\_FK** INTEGER,  
**CONSTRAINT** PK\_LOGONS PRIMARY KEY(LOGON\_ID));
- **CREATE MEMORY TABLE MEETINGS** (**MEETING\_ID** INTEGER NOT NULL,  
**CIPHER\_LENGTH** INTEGER, **NAME** VARCHAR(256), **EXTERNAL\_CONTEXT** VARCHAR(256),  
**SCHEDULED\_START** VARCHAR(256), **DESCRIPTION** VARCHAR(256), **MEETING\_KEY** VARCHAR(256),  
**MODERATOR\_KEY** VARCHAR(256), **DELETED** BOOLEAN NOT NULL, **SANDBOX\_CLIENT** BOOLEAN NOT NULL,  
**EMBEDDED** BOOLEAN NOT NULL, **PUBLISH** BOOLEAN NOT NULL, **DONE\_PAGE** VARCHAR(256),  
**MEETING\_START** TIMESTAMP, **MEETING\_END** TIMESTAMP, **SCRIPT\_ID\_FK** INTEGER, **CUSTOMER\_ID\_FK** INTEGER,  
**CONSTRAINT** PK\_MEETINGS PRIMARY KEY(MEETING\_ID), **CONSTRAINT** FK\_MEETINGS\_SCRIPT FOREIGN KEY(SCRIPT\_ID\_FK)  
REFERENCES SCRIPTS(SCRIPT\_ID), **CONSTRAINT** FK\_MEETINGS\_CUSTOMER FOREIGN KEY(CUSTOMER\_ID\_FK) REFERENCES CUS-  
TOMERS(CUSTOMER\_ID));
- **CREATE MEMORY TABLE PARTICIPATIONS** (**PARTICIPATION\_ID** VARCHAR(256) NOT NULL,  
**LOGON\_NAME** VARCHAR(256), **EMAIL** VARCHAR(256), **IP\_ADDR** VARCHAR(256), **JAVA\_VENDOR** VARCHAR(256),  
**JAVA\_VERSION** VARCHAR(256), **OS\_NAME** VARCHAR(256), **OS\_VERSION** VARCHAR(256), **OS\_ARCH** VARCHAR(256),  
**BEGINTIME** TIMESTAMP, **ENDTIME** TIMESTAMP, **MEETING\_ID\_FK** INTEGER,  
**CONSTRAINT** PK\_PARTICIPATIONS PRIMARY KEY(PARTICIPATION\_ID),  
**CONSTRAINT** FK\_PARTICIPATIONS\_MEETING FOREIGN KEY(MEETING\_ID\_FK) REFERENCES  
MEETINGS(MEETING\_ID));
- **CREATE MEMORY TABLE SCRIPTS** (**SCRIPT\_ID** INTEGER NOT NULL, **NAME** VARCHAR(256),  
**DESCRIPTION** VARCHAR(256), **CUSTOMER\_ID\_FK** INTEGER,  
**CONSTRAINT** PK\_SCRIPTS PRIMARY KEY(SCRIPT\_ID), **CONSTRAINT** FK\_SCRIPTS\_CUSTOMER FOREIGN  
KEY(CUSTOMER\_ID\_FK) REFERENCES CUSTOMERS(CUSTOMER\_ID));
- **CREATE MEMORY TABLE CONTENTSLIDES** (**CONTENTSLIDE\_ID** INTEGER NOT NULL, **SLIDE\_SIZE** INTEGER NOT NULL,  
**SLIDE\_NAME** VARCHAR(256), **SLIDE\_DESCRIPTION** VARCHAR(256), **CONTENTUPLOAD\_ID\_FK** INTEGER,  
**CONSTRAINT** PK\_CONTENTSLIDES PRIMARY KEY(CONTENTSLIDE\_ID)).

## C. Interface to create new ad hoc conference

At figure 36 it can be seen the interface, of *Tagarela*, to create a new ad hoc conference. It is also marked the three fields that are different to create this type of conference, comparing with the thematic conferences.

The screenshot shows the 'New Service - Enterprise PT' interface in Tagarela. The header includes the Tagarela logo, 'Inovação' text, and '10051580 | Exit' and 'Help/FAQ Contact' links. Navigation tabs are 'Conferences', 'Admin', 'New Conference', and 'New Service'. The main form has the following fields and options:

- Chat Room Name:
- SIP Extension:  Username:  Password:
- Maximum number of simultaneous sessions:  (highlighted with a red box)
- Available Resources:  Audio,  Video,  Chat,  File Sharing,  Desktop sharing,  Slideshow sharing
- Max number of Participants:  (highlighted with a red box)
- Max Duration (minutes):  (highlighted with a red box)
- Room's Department:
- Subscribers:  (Selected)
- Subscribers by Department:
- Subscribers list:  (Other...)
- Buttons: '<- Add one by one', '<- Add all', 'Create Conference'

(c) Portugal Telecom Inovação, SA

Figure 36: Interface to create new ad hoc conference.

## D. Description of the *Floor Control* area

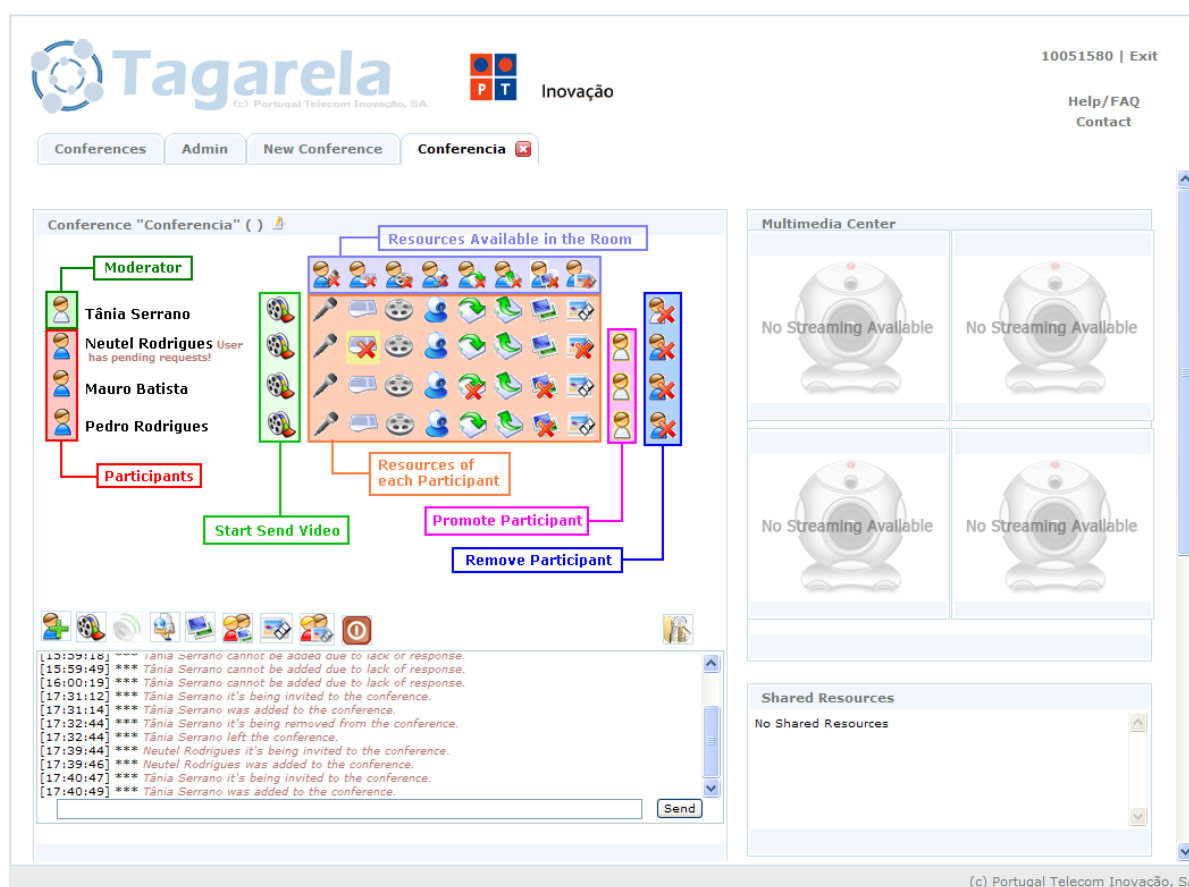











Figure 37: *Tagarela* Floor Control.

In the *Floor Control* area the participants have available a set of buttons that provides interaction with *Tagarela*. This buttons are the following:

- invite a user to join the meeting; 
- share files; 
- start a desktop sharing; 
- join a desktop sharing; 
- start a slideshow sharing; 
- join a slideshow sharing; 
- exit the meeting; 
- show/hide the interface to manage the slides used for the slideshow sharing. 



## E. Start Desktop Sharing

Once inside the conference, a participant can decide to start one desktop sharing. To do this, it is only required that the room has the desktop sharing resource available, and to click in the "Start Desktop Sharing" button, at the *Floor Control* area. 

The participant may not have permission to start a desktop sharing. In this case, when he clicks in the "Start Desktop Sharing" button, it will be sent one permission request to the *ModeratorConf*, and the participant will receive the next message: "Your request has been successfully sent. It may take a few seconds to process". If there is already one desktop sharing underway, at the conference room, then the participant will receive the next message: "There is a desktop sharing".

In the case that the user has permission to start a desktop sharing, and there is no sharing of this type underway, then a new *tab* will be open with the content represented in figure 38.

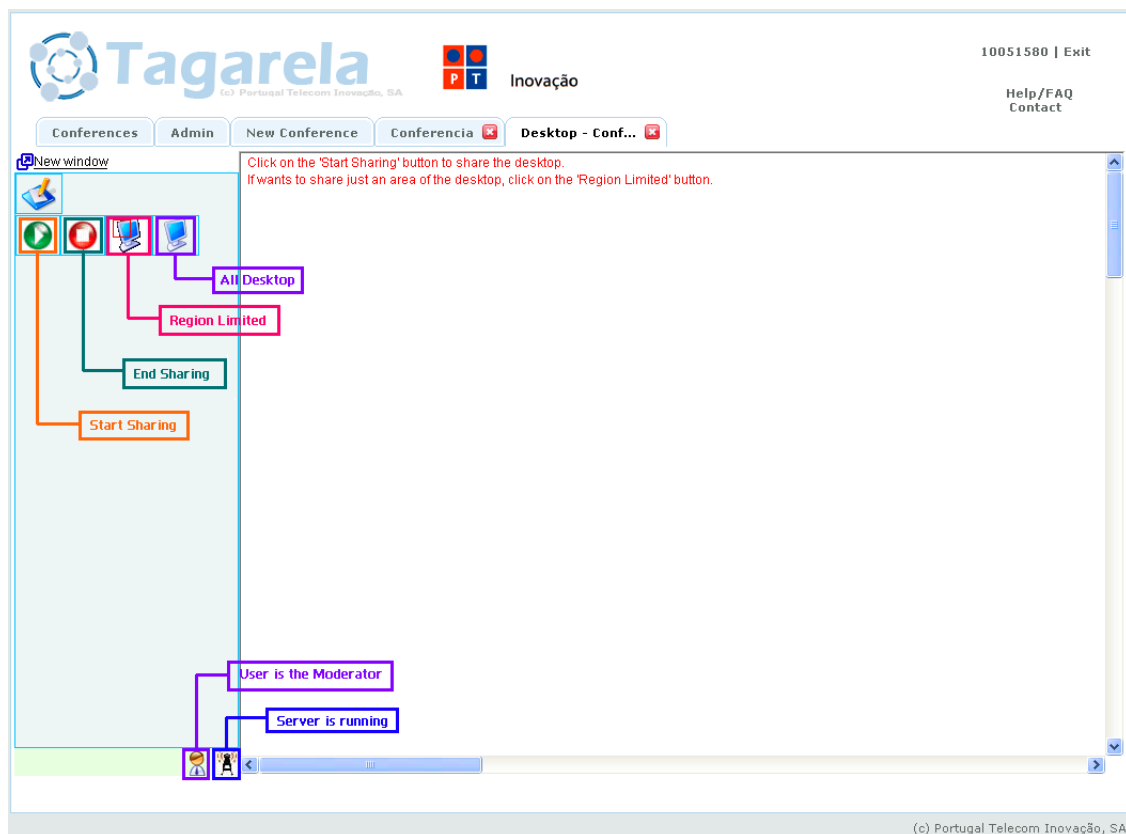



Figure 38: Interface of a desktop sharing, with the description of the available buttons.

## F. Start Slideshow Sharing

When the participant is inside the conference, he can decide to start one slideshow sharing. To do this, it is only required that the room has the slideshow sharing resource available, and to click in the "Start Slideshow Sharing" button, at the *Floor Control* area. 

The participant may not have permission to start a slideshow sharing. In this case, when he clicks in the "Start Slideshow Sharing" button, it will be sent one permission request to the *ModeratorConf*, and the participant will receive the next message: "Your request has been successfully sent. It may take a few seconds to process". If there is already one slideshow sharing underway, at the conference room, then the participant will receive the next message: "There is a slideshow sharing".

In the case that the user has permission to start a slideshow sharing, and there is no sharing of this type underway, then a new *tab* will be open with the content represented in figure 39.

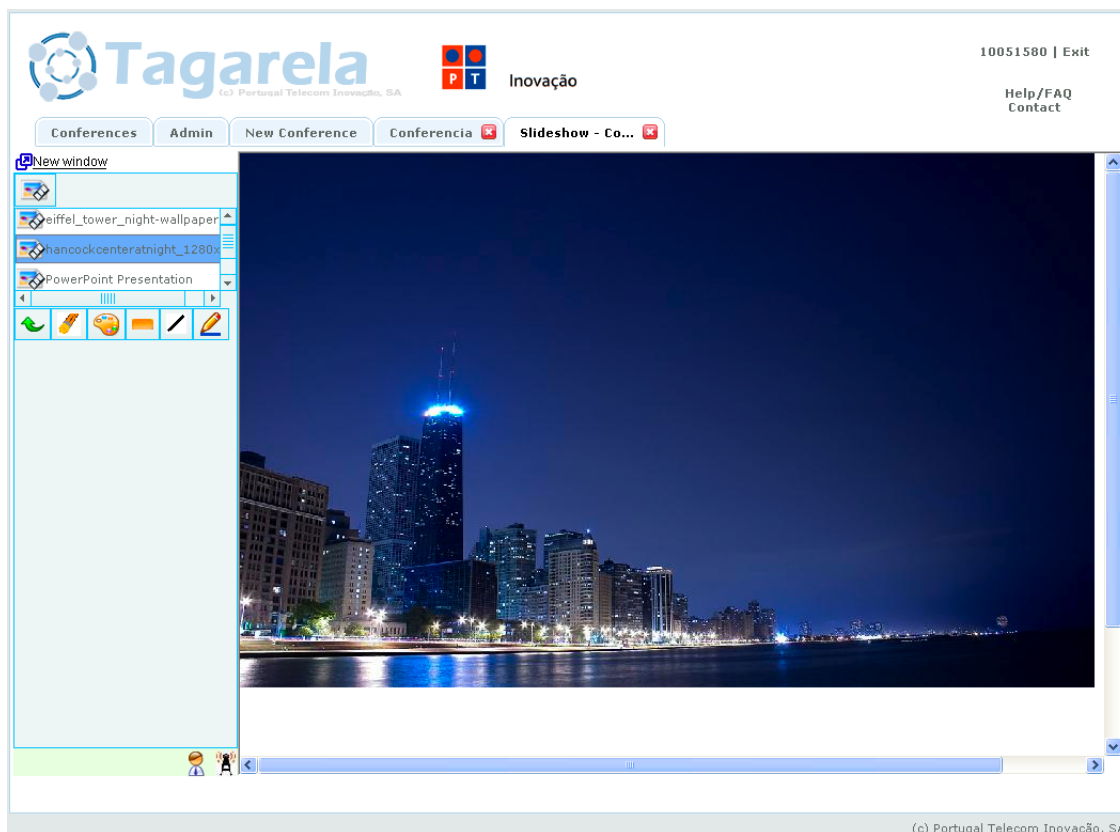


Figure 39: Example of a slideshow sharing page.

## G. Additional Information to Join a Desktop/Slideshow Sharing

To join the sharing, the participant only needs to click in the "Join Desktop Sharing" button (👤) or in the "Join Slideshow Sharing" button (👤), that there is at the *Floor Control* area.

When a participant wants to join a sharing and there is no sharing underway, then the participant will receive a message to explain it. It will be: "There isn't any desktop sharing" or "There isn't any slideshow sharing". If, on the other way, the participant is already inside the desktop/slideshow sharing, of the actual conference room, then he/she will receive one of the following message, depending the type of the sharing wanted: "User is already in this desktop sharing" or "User is already in this slideshow sharing".

In the case that the user can join a desktop sharing that is underway, then a new *tab* will be open with the content represented at figure 40. In this example, the *ModeratorWH* is only showing only a part of his desktop, and not the all desktop. Because of this, the user only sees the area inside the green rectangle.

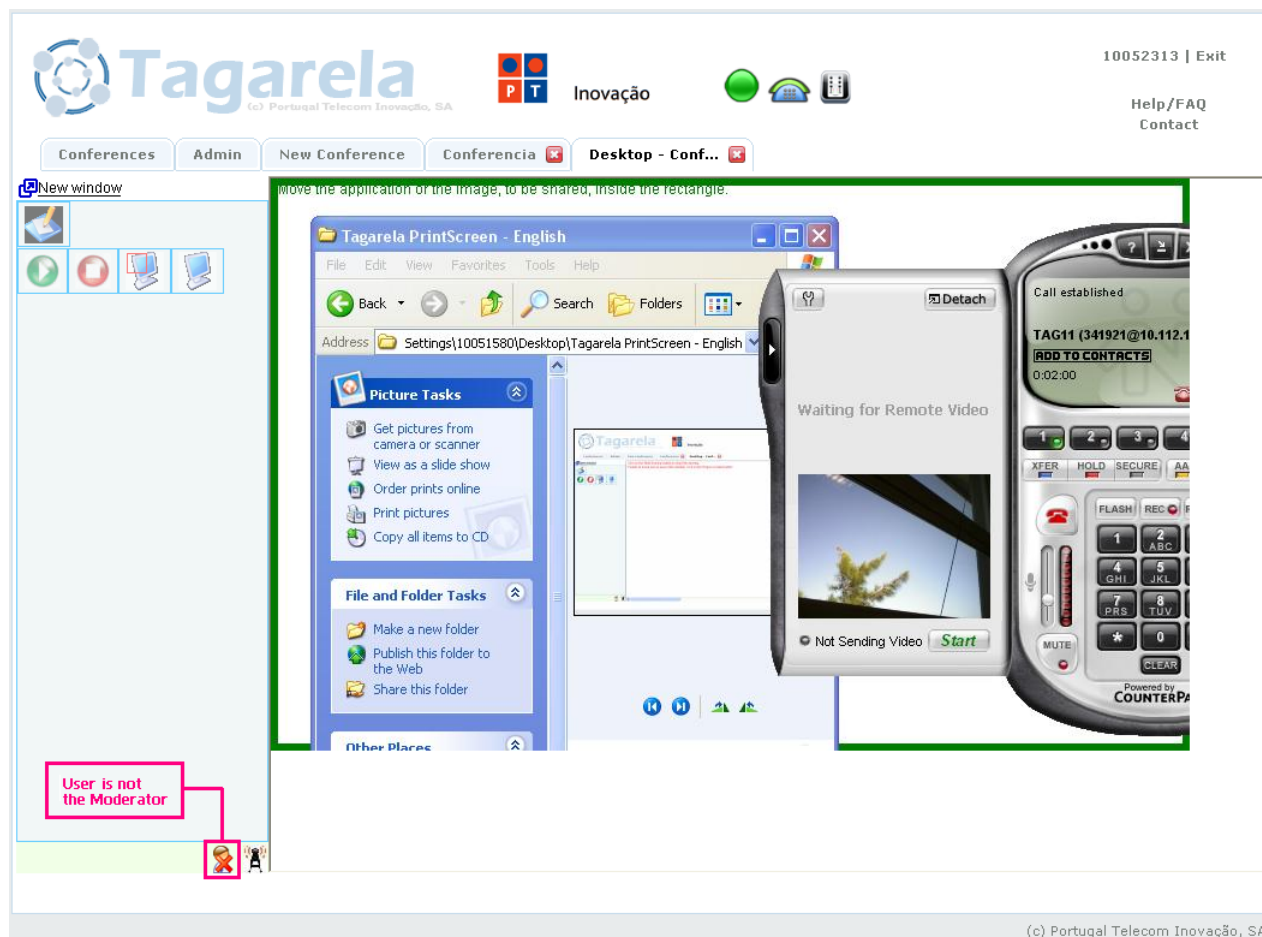


Figure 40: Example of the page that the user, who joined a desktop sharing, will see.

In the case that the user can join a slideshow sharing that is underway, then a new *tab* will be open with the content represented at figure 41. In this example, the *ModeratorWH* is showing an image where he/she has already made some changes in it, using the tools available by the *WebHuddle*.

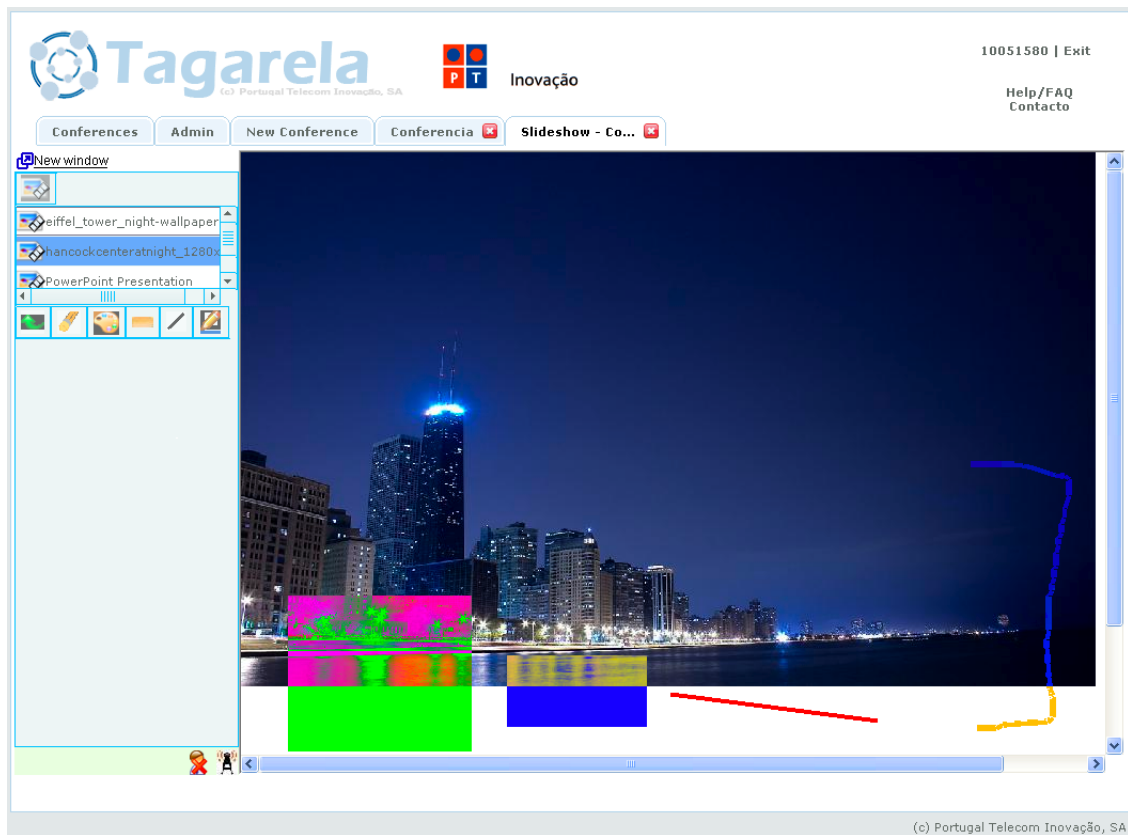



Figure 41: Example of the page that the user, who joined a slideshow sharing, will see.

## H. More about Upload Files to a Slideshow

When a participant clicks in the "Manage Slideshow Files" button (  ), at the *Floor Control* area, it show or hide an area where the participant can upload or manage files to the slideshow sharing (see figure 42).

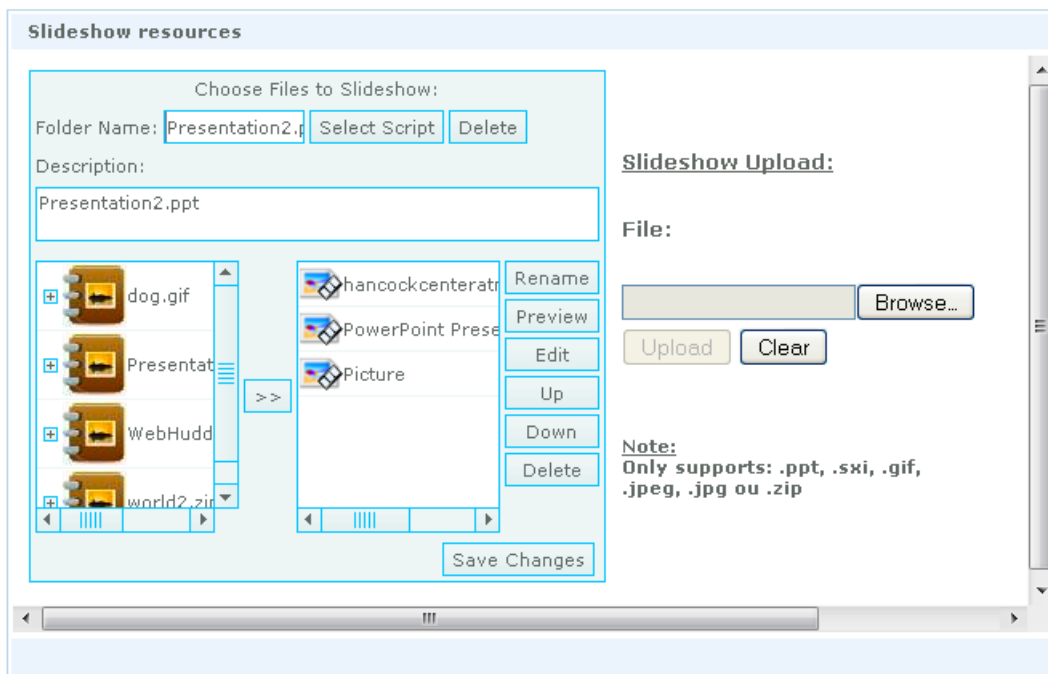


Figure 42: Interface to upload and manage the slides to use in a slideshow sharing.