NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

DIOGO DANIEL ANDRADE NUNES DOS SANTOS

Bachelor of Computer Science

PROOF OF CONCEPT OF A NEW AUDIO DISPATCHER

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon

September, 2021

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

PROOF OF CONCEPT OF A NEW AUDIO DISPATCHER

DIOGO DANIEL ANDRADE NUNES DOS SANTOS

Bachelor of Computer Science

Adviser: Rui Lourenço
Product Manager, Thales Portugal

Co-adviser: Matthias Knorr
Assistant Professor, NOVA University Lisbon

Examination Committee

Chair: Margarida Mamede
Assistant Professor, NOVA University Lisbon

Rapporteur: Sofia Cavaco
Assistant Professor, NOVA University Lisbon

Member: Rui Lourenço
Product Manager, Thales Portugal

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
September, 2021

Proof of concept of a new Audio Dispatcher

This document was created using the (pdf)LaTeX processor, based on the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

This thesis would not have been possible without the kind support and help of many individuals.

To my advisors Matthias Knorr and Rui Lourenço for their constant support and availability, as well as all the knowledge and ideas they passed on to me that helped make this thesis possible.

To my AD teammates, Pedro Luz, João Brito and Rui Lourenço, who helped and instructed me when I needed guidance and assistance.

To my father, for his extraordinary motivation and encouragement, not only during this journey but in my entire life.

To my mother, for her unconditional support and for always believing in me.

To my brother, for always being by my side since the beginning.

Finally, I want to thank the rest of my family and friends for all their support on this long and difficult path.

Abstract

To excel in today's world, rail controllers need to be able to communicate with numerous personnel across different mobile and fixed networks, in order to make well-informed decisions effortlessly.

Audio Dispatcher is a software product that allows the railway control center operators to manage different voice communication systems in an integrated manner, by being able to concentrate them in a single Web graphic interface. It enables operational communication between control centers, train drivers, shunting personnel, maintenance personnel, neighbourhood dispatchers and local train stations.

Audio Dispatcher already had two previous versions, however, the second version had some problems and was not according to Thales GTS division latest technological stack. This new version of the Audio Dispatcher is integrated into a multi-platform framework named GDP UI Toolkit whose main advantage is that it facilitates the rapid development of applications based on a predefined toolbox of components and microservices.

In the course of this product's renewal, new functionalities need to be added, such as multimedia file sharing, chat sessions and video calls. This 3^{rd} version also intends to make the product more versatile, easy to set up and configure.

The AD product required a new and well-defined interface. This is the main purpose and the problem this dissertation was proposed to solve, in doing so, we developed the HMI for the new Audio Dispatcher system, focusing on two central modules: the **Call Manager** and the **Contact Manager**.

Keywords: Audio Dispatcher, Call Manager, Contact Manager, HMI, GDP, Widget, SIP, PABX Asterisk.

Resumo

Para se destacar no mundo de hoje, os controladores ferroviários necessitam de comunicar com várias pessoas em diferentes redes móveis e fixas, para tomar decisões bem informadas sem qualquer esforço.

O **Audio Dispatcher** é um produto de software que permite aos operadores de centros de controlo ferroviário giram numa maneira integrada diferentes sistemas de comunicação de voz, podendo concentrar esses sistemas numa única interface gráfica Web. Este software permite a comunicação operacional entre centros de controlo, maquinistas, pessoal de manutenção e estações de comboio locais.

O **Audio Dispatcher** já tinha duas versões anteriores, porém, a segunda versão apresentava alguns problemas e não estava de acordo com a exigência tecnológica da divisão GTS da Thales. Esta nova versão do Audio Dispatcher está integrada numa framework multiplataforma denominada GDP UI Toolkit, cuja principal vantagem é que facilita o rápido desenvolvimento de aplicações com base numa coleção predefinida de componentes e microsserviços.

No decorrer da renovação deste produto, novas funcionalidades serão adicionadas, como a partilha de ficheiros multimédia, chat e vídeo chamadas. Esta 3ª versão pretende também tornar o produto mais versátil, fácil de instalar e configurar.

O produto AD precisava de uma nova e bem definida interface. Este é o principal objetivo e o problema que esta dissertação foi proposta a resolver, ao fazê-lo, desenvolvemos a HMI para o novo sistema Audio Dispatcher, com o foco em dois módulos centrais: o **Call Manager** e o **Contact Manager**.

Palavras-chave: Audio Dispatcher, Call Manager, Contact Manager, HMI, GDP, Widget, SIP, PABX Asterisk.

Contents

List of Figures

Acronyms

AD Audio Dispatcher

API Application Programming Interface

GDP Ground Transportation Digital Platform

GIS Geographic Information System

GSM-R Global System for Mobile Communications – Railway

GTS Ground Transportation Systems

HMI Human-Machine Interface

IP Internet Protocol

IVVQ Integration Validation Verification Qualification

LDAP Lightweight Directory Access Protocol

NTP Network Time Protocol

OCC Operation Control Centre

P2P Peer to Peer

PABX Private Automatic Branch Exchange

PSTN Public Switched Telephone Network

REST Representational State Transfer

SIP Session Initiation Protocol

TETRA Terrestrial Trunked Radio

VoIP Voice over Internet Protocol

VRS Voice Recording System

Chapter 1

Introduction

1.1 Context and Motivation

Nowadays, train dispatchers need to be able to communicate with multiple associates in the railway network across different mobile and fixed networks. With the improvement of the technology used in the railway systems, new problems emerged in the communications of the railway world. This is where the Audio Dispatcher comes in, to try to help with these problems.

Audio Dispatcher is an integrated communication tool that allows operators of a metro/rail control center to control an integrated form of voice communications systems (analog, IP, radio, GSM-R, TETRA), in one unified console with a graphical interface based on a web browser.

The version currently under development is the 3^{rd} generation of the Audio Dispatcher product. The first release (v1) was created back in 2007 when Thales created a product to manage in a unified way, different types of communications that Refer Lisbon OCC operators needed to handle. It was called CCI (Consola de Comunicações Integrada) and it is still in use today. Its second release (v2) was created in 2016, to be part of the Thales Integrated Supervision and Communication Solution (ISCS) framework. This version was a refactor from version 1, using a different technological stack (Java based), with a web-based HMI. Despite the refactoring, its goals remained essentially the same. The third release (v3) is now being developed. It intends to make the product more versatile, easy to set up and configure. It uses a distributed architecture based on microservices. Its technical architecture is being supported by the brand new Thales GDP framework.

The following factors contributed for Thales to refactor the previously existing version of the Audio Dispatcher product to a new version:

- Winning bid competitiveness;
- Distinct, non-integrated product landscape, as it was not possible to put together into a single unified operational console several telephony devices and interfaces across different mobile and fixed networks;
- Customers demanding state-of-the-art products. The existing version of the product was not according to Thales GTS division latest technological stack, as it was unable to address new business opportunities like modern Unified Communication Solutions;
- Poor adaptation and flexibility, leading to long-drawn delivery cycles and tedious configuration.

The new Audio Dispatcher is being designed to be a configurable generic software product for managing voice communications in an integrated manner. All the needed mechanisms, functionalities and interfaces must be present within the product. The customer-specific parts, which include interface appearance, color schemes, 3^{rd} party drivers, concrete textual content, particular workflows and so on, and also must be as configurable as possible.

Audio Dispatcher distinguishes itself from a standard Telephony solution by enabling customers to define and customize telephony operational rules capable of managing and handling calls from different devices and systems such as help points, train driver consoles, office calls, calls from external stakeholders (e.g., security forces, city council and other regulatory entities) and Radio based telephony devices.

1.2 Objectives and Contributions

The main goal of this Thesis was the implementation of the Human-Machine Interface (HMI) for the latest version of the Audio Dispatcher Product.

This was done using the GDP UI toolkit, which is the common integration framework used in the front-end development of Thales products.

In this implementation, I was responsible for developing the front end of this application which consisted of two complex Web Components. The first one is the Call Manager. This widget is the main view of the application, where the users are able to perform most actions regarding the call management part. The second widget is the Contact Manager, which focuses on the contact management part, where the users are free to organize their contact list.

My biggest contribution to this project was to overcome the main challenge of integrating all the initially planned features for the AD into the new Thales platform, GDP, which was achieved with great success, reaching all the goals set by the AD team along with the feedback from their customers.

1.3 Document Structure

This document is organized in the following way:

- **Chapter 1: Introduction** - In this chapter, we provide the context of the product and its main motivations, following by, this dissertation's objectives.
- **Chapter 2: Problem Analysis** - Here we will explain the product's primary objectives. After that, we will introduce the Product's requirements. Finally, we will present some similar products and our product's key differentiators regarding the competition.
- **Chapter 3: State of the Art** - In chapter 3, we will summarize the research done regarding the state-of-the-art for the technologies used.
- **Chapter 4: Processes and Methodologies** - In chapter 4 is described the methodologies that Thales adopted on their product development, followed by, the tools used to help with the team's management.
- **Chapter 5: Project / Approach** - Here we will elaborate on the Audio Dispatcher solution, starting with the project's Architecture, then the Middleware and Database. And ending with a report of the frontend development followed by the finished product.
- **Chapter 6: Verification and Validation** - In this chapter we will describe the tests that were performed regarding the application, and the strategy that the IVVQ team will follow for our product's testing.
- **Chapter 7: Conclusion** - Finally we will list the conclusions attained with the realization of this Thesis. And in the end, we will present a brief outlook on the Project's future.

Chapter 2

Problem Analysis

In this chapter we will try to explain the main purposes of the product in order to give a better context of its importance and place in today's market. We will start by enumerating the project's most important objectives, followed by a brief description of the application roles and profiles, while also listing a set of permissions to be assigned to those roles. Furthermore, we will present the project's user requirements, succeeded by use cases diagrams, where we can see the user's possible interactions with our application.

Some of these resources were obtained from internal Thales documents prepared by the AD team, for example, the system requirements and operational scenarios, presented in the sections below.

In the end, there will be an analysis of Audio Dispatcher's similar solutions in the railway industry, followed by the product's key differences in contrast to its competition.

2.1 Technical Objectives

The main technical objectives assigned to the Audio Dispatcher Project (System) are:

- Concentrate in a single workstation dedicated HMI all operational voice, text and video communications for railway operators;
- Enable operational communication between command and control centers, train drivers, shunting personnel, maintenance personnel, neighbourhood dispatchers and local train stations;
- Manage contacts from different stakeholders into one single console;
- Allow to define custom operational rules for handling calls;
- Manage different calls priorities;
- Support SIP VoIP calls and standard PABX technology.

2.2 Roles and Profiles

Control centers must be capable of communicating with numerous stakeholders across different mobile and fixed networks, and make well-informed decisions effortlessly. The Audio Dispatcher optimizes communication between these control centers, train drivers, maintenance staff, local train stations and Help Points (passengers).

We chose some default profiles for the AD system, in order to emulate the clients needs, for example, AD Operator, AD Administrator, AD Supervisor, Maintenance, Train Drivers, Passengers, etc.

The main profiles used in our system are the Operator, Administrator and Supervisor. A dedicated tool, called GDP User Manager, is being used to associate these profiles to specific functionalities and permissions developed inside the AD product.

Audio Dispatcher shall support at least the following permissions, while remaining open to more additions:

- **AdCallMgrReadWrite:** Mandatory. It is always necessary to make or receive calls;
- **AdCallHistoryRead:** Necessary if the user needs to have access to Call History information in read only mode (for consultation purposes only);
- **AdCallHistoryReadWrite:** Necessary if the user needs to have access to Call History information and can add comments to existing call detailed records or needs to export any information from the system to external media devices;
- **AdContactMgrRead:** Necessary if the user needs to have access to Contact Manager in order to consult contacts and contact filters related information;
- **AdContactMgrReadWrite:** Necessary if the user needs to have access to Contact Manager and needs to add, update or delete contacts and contact filters related information and records;
- **AdSystemSettingsRead:** Necessary if the user needs to have access in read only mode to System Settings in order to consult Audio Dispatcher configuration parameters, such as PABX servers' settings and 3rd party servers connection settings;
- **AdSystemSettingsReadWrite:** Necessary if the user needs to have access to System Settings in order to add, update and delete Audio Dispatcher configuration parameters, such as PABX servers' settings and 3rd party servers connection settings;
- **AdDBWorkerReadWrite:** Mandatory. It is always necessary to register the calls that were made;

- **AdDriverMgrReadWrite:** Mandatory: It is always necessary to establish a connection to the telephony server;
- **LogMgrRead:** Only necessary if the user needs to have access to the GDP Log Manager module and to consult and inspect Audio Dispatcher technical logs, as well as from other applications;
- **LogMgrReadWrite:** Only necessary if the user needs to have access to the GDP Log Manager module and to add comments to any of the existing Audio Dispatcher technical logs as well, as well as from other applications;
- **UsrMgrRead:** Necessary to access to the GDP User Manager application, where users and roles are created and managed. The Read permission means that the user is only able to consult information and cannot add or update any existing information;
- **UsrMgrReadWrite:** Necessary to access to the GDP User Manager application where users and roles are created and managed. The ReadWrite permission means that the user is only able to change, update or add information in the existing users or roles.

2.3 User Requirements

To better understand and contextualize the work I performed, there are several requirements and use cases that we need to preview. These requirements and use cases are available in a dedicated requirements management repository tool, named DOORS [9], where all product requirements are identified, detailed, organized and traced to test modules where specific test cases are detailed and assigned to each of the requirements described.

According to Thales SW development methodology, these requirements are then exported to a dedicated document (System Subsystem Specification - SSS [17]). The section below will present the requirements and use cases of the AD product.

Starting by the functional requirements, the original list is comprised of all the requirements of the features planned to be developed in a long-term plan. Considering this list contained an extensive number of these requirements, we will present a filtered list of the functional requirements that were important for my work.

In the Table 2.1, will be presented the Peer to Peer (P2P) calls' operations, the following table will display the Conference calls' operations (Table 2.2). In the Table 2.3 are the rest of the Call Management features. In the end of the functional requirements, are the tables 2.4 and 2.5, detailing the Call History and Contact Management requirements, respectively.

These tables will be followed by the non-functional requirements of the product (Tables 2.6, 2.7 and 2.8). Here are listed the product's requirements affecting Usability, Performance and Security, respectively.

2.3.1 Functional Requirements

ID	Name	Description
0051	Create P2P Call	The AD system shall allow the operator to make P2P calls to other operators or external entities.
0052	Create Video Call	The AD system shall allow the operator to perform Video Calls.
0053	Create Call with Location	The AD system shall allow the operator to make a call using a location on a map layer (e.g., Help Point).
0061	Incoming P2P Call	The AD system shall provide the operators with a notification warning when an incoming call gets in, by displaying it in the on-going call queue list.
0066	Answer P2P Call	The AD system shall allow the operator to answer an incoming call.
0067	Automated Call distribution	The AD system shall automatically distribute calls to different operating locations based on business rules such as the hunting mode algorithm.
0068	Answer Call in Queue	The AD system shall allow the operator to retrieve a call from a queue in order to answer it, even if the call is not on top of the answering queue.
0069	Remote Call Pickup	The AD system shall allow the operator to answer a call from another AD operating position.
0076	Hang-up P2P Call	The AD system shall allow the operator to hang-up a P2P call.
0081	Reject P2P Call	The AD system shall allow the operator to reject a P2P call.
0101	Hold P2P Call	The AD system shall allow the operator to put a P2P call on hold.
0111	Unhold P2P Call	The AD system shall allow the operator to unhold a P2P call.
0121	Mute P2P Call	The AD system shall allow the operator to mute an ongoing P2P call.
0131	Unmute P2P Call	The AD system shall allow the operator to unmute an ongoing P2P call.
0141	Attended Transfer P2P Call	The AD system shall allow the operator to perform an attended call transfer to another operator.
0142	Blind Transfer P2P Call	The AD system shall allow the operator to do a blind call transfer to another operator.
0151	Volume control	The AD system shall allow the operator to control the volume level of a call.
0161	Redial	The AD system shall allow the operator to redial from a recent calls list.
0171	Recording Calls	The AD system shall provide the possibility to record calls by using a dedicated Voice Recording System.

Table 2.1: Functional Requirements - P2P Calls Operations

ID	Name	Description
0196	Create Conference Call	The AD system shall allow the operator to create a conference call using a list of contacts available in the contact list.
0206	Answer Conference Call	The AD system shall allow the operator to join a conference call.
0211	Cancel Conference Call creation	The AD system shall provide the conference call moderator with the possibility to cancel the creation of a conference call before it is answered by the destination peers.
0215	Add Participant Conference Call	The AD system shall allow the conference call moderator to add another participant to the conference call.
0221	Moderator Hang-up Conference Call	The AD system shall provide the moderator in a conference call the possibility to hang-up a call to: all its participants, a specific participant or himself.
0222	Participant Hang-up Conference Call	The AD system shall provide any participant in a conference call the possibility to hang-up from the call.
0231	Reject Conference Call	The AD system shall allow the operator to reject joining a conference call.
0241	Hold Conference Call	The AD system shall allow the conference call moderator to: • Set the conference call on hold to all its participants. • Individually set a participant in the conference call on hold. • Put himself on hold.
0251	Unhold Conference Call	The AD system shall allow the conference call moderator to: • Unhold the conference call to all its participants. • Individually unhold any of the participants of the conference call. • Unhold himself in the conference call.
0261	Moderator Mute Conference Call	The AD system shall allow the conference call moderator to: • Mute himself in a conference call. • Individually mute any participant in a conference call. • Mute all the participants in the conference call.
0262	Participant Mute Conference Call	The AD system shall provide any participant in a conference call the possibility to mute himself in the call.
0271	Moderator Unmute Conference Call	The AD system shall allow the conference call moderator to: • Unmute himself in a conference call. • Individually unmute any participant in a conference call.
0272	Participant Unmute Conference Call	The AD system shall provide any participant in a conference call the possibility to unmute himself in the call.
0281	Volume Control Conference Call	The AD system shall provide the conference call moderator with the possibility to adjust the call volume level of other participants in the conference call.

Table 2.2: Functional Requirements - Conference Calls Operations

ID	Name	Description
0411	Call Snooping	The AD system shall provide the operator with the proper role, the possibility to wire a call from other operators without them notice that he/she is listening to the call.
0412	Call Whisper	The AD system shall provide the operator with the proper role, the possibility to wire a call from other operators and at the same time, talk (whisper) to one of the operators in the call without the others know about it.
0421	Call Priority Management	The AD system shall be able to process calls based on the call priority attribute (e.g., emergency, information, etc).
0431	Call Queues	The AD system shall be able to support call-queue management. This shall provide the system the capability to manage call queues and distribute the calls to the responsible operators.
0432	Call Queues (Priority)	The AD system shall be able to process the calls in the queues based on their priorities. For instance, emergency calls shall have higher priority than information calls and will be represented higher in the call management queue.
0442	Call Status	The AD system shall allow the operator to monitor the status of each of the active calls displayed in the call queue monitoring area (e.g., call is ringing, call is on hold).
0445	Active Call	The AD system shall allow the operator to monitor the information of the active call (e.g., parties in the call and in what mode (i.e., muted, on hold, etc)).
0461	Merge Call	The AD system shall allow the operator to merge calls.
0471	Divert Call	The AD system shall allow the operator to automatically divert a call.
0541	Dial Pad	The AD system shall allow the operator to use a dial pad in the AD HMI to make and control calls.

Table 2.3: Functional Requirements - Call Management

ID	Name	Description
0521	Call History Records	The AD system shall allow the operator to visualize a list of all the calls and short text messages that he/she made or received during a predefined period of time.
0522	Call History Ordered	The AD system shall allow the operator to visualize a list of the call history records ordered by time, caller name or extension.
0523	Call History Notes	The AD system shall allow the operator to add notes into each of the call records displayed in the call history.
0524	Call History Export	The AD system shall provide the operator with the possibility to export the call history records to a .csv file.
0525	Call History Details	The AD system shall allow the operator to visualize the following call history details: • Date/Time • Duration • Call notes • Creator • Participants • Priority • Call Type

Table 2.4: Functional Requirements - Call History

ID	Name	Description
0531	Manage List of Contacts	The AD system shall allow the operator to arrange and manage (add, edit or delete) a list of contacts.
0532	Search	The AD system shall allow the operator to search the list of contacts by extension, name, etc.
0533	Favorites	The AD system shall provide the operator with a list of favorite contacts, where the operator can add or remove contacts to this list.
0534	Create Call from the List of Contacts	The AD system shall allow the operator to select several contacts from the contacts list and initiate a P2P call or a conference call.
0535	Priority	The AD system shall allow the operator to visually identify the priority of the contacts in the contact list.
0571	Device Status	The AD system shall display to the operator the status (e.g., busy, online, offline, etc.) of other AD operators in their list of contacts.
0572	Device Status in Real Time	The AD system shall provide the operator with real time refresh states on the HMI of the telephony devices status in the system.

Table 2.5: Functional Requirements - Contact Management

2.3.2 Non-Functional Requirements

ID	Name	Description
0656	Support Different Languages	The AD system shall be designed to support multiple languages.
0660	Operator Logs	The AD shall log operator actions performed in the HMI. These operator actions shall be made available for consultation in the AD HMI, based on the operator profile.
0661	AD HMI User Info	The AD HMI shall make visible to the operator, in case of successful login, the user name and current operator extension.
0662	AD HMI Help	The AD HMI shall provide the operator integrated online help for all the system functions.

Table 2.6: Non-Functional Requirements - Usability

ID	Name	Description
0670	Capacity	The AD system shall operate, monitor and control the required communication devices and SIP extensions to at least 10.000 extensions.
0672	Scalability	The AD system shall be scalable and rely on a redundant IP-network design to comply with emergency call system needs.
0687	Reliability	The AD system shall be able to automatically resume communication to the AD server in case of communication failure.
0696	Availability	The AD system availability rate shall not be less than 99%. This number may vary depending on project characteristics. The AD system shall be designed to have service continuity in the presence of central server failure, PABX failure or any other communication device failure.
0706	Stability	The AD system shall be designed to prevent failures or breakdowns due to invalid inputs. The AD system shall be able to restore the operation from fault condition.
0708	Maintainability	The AD system shall be suitably designed to minimize the need for frequent preventive maintenance.

Table 2.7: Non-Functional Requirements - Performance

ID	Name	Description
0716	Security	The AD system shall be able to resist unauthorized attempts at usage (hacking attempts by external subjects). The AD system shall be able to continue providing service to legitimate users while under denial of service attack (resistance to DoS attacks). The AD system shall be cyber secured according to the ISA 62443 SL2 standard.
0721	Secured Deployment Servers	It is assumed that all Audio Dispatcher related software modules are to be deployed in servers with restricted access permissions, where only authorized maintenance personal can access.
0726	Secured Network	It is assumed that the communication between all Audio Dispatcher modules (e.g. Audio Dispatcher Core, Audio Dispatcher HMI) and external systems (e.g. TMS, SCADA, Time Server, PA Server, etc.) is performed over a secured, private network.

Table 2.8: Non-Functional Requirements - Security

2.3.3 Use cases

This section presents two use case diagrams, one for the call management (Fig. 2.1) and one for contact management (Fig. 2.2). All the functionalities presented in these use cases correspond to at least one requirement. These use cases demonstrate the interactions between the different roles and the system in a summarized and visual way. After the diagrams, there will be presented some Audio Dispatcher operational scenarios based on a few use cases.

2.3.3.1 Use Case Diagrams

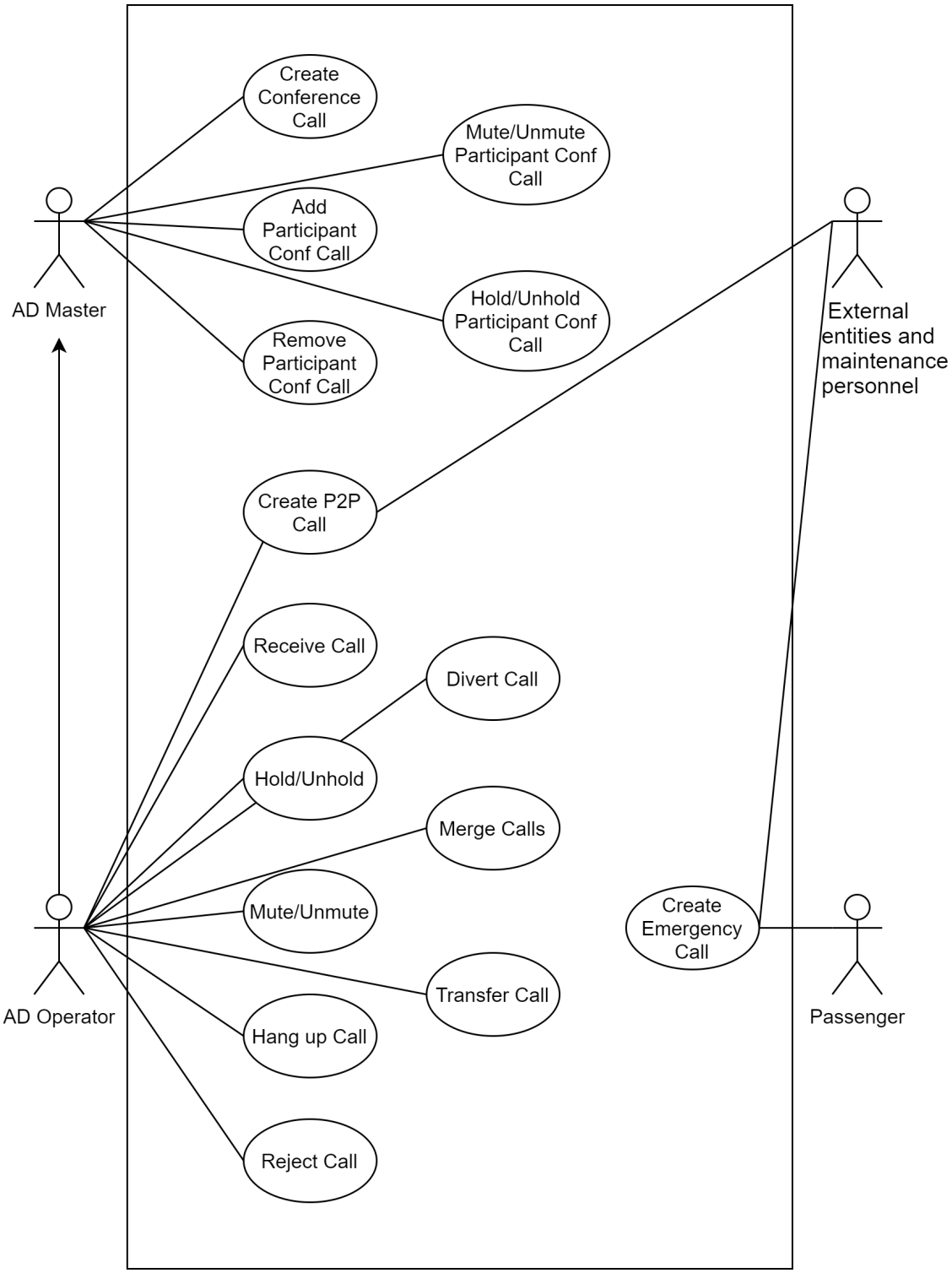


Figure 2.1: Use Case Diagram - Call Management

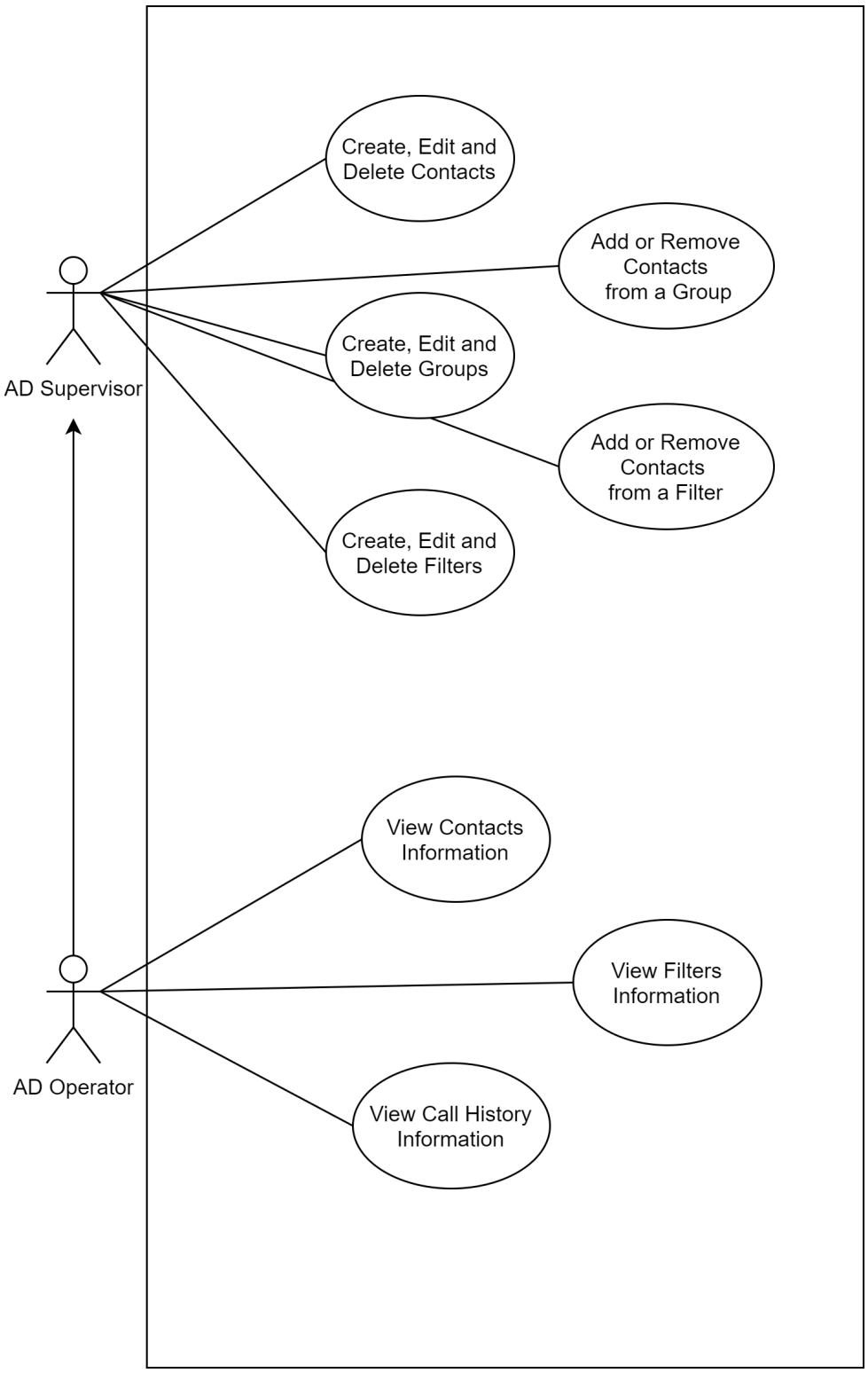


Figure 2.2: Use Case Diagram - Contact Management

2.3.3.2 Operational Scenarios

In this section, we will present some operational scenarios that our application may encounter in the future. These tables were extracted from the Audio Dispatcher SSDD internal document [3]. These tables are divided into three columns, the first consists of the actors that participate in the scenario, followed by the action to be performed in our application. In the last column, it will be specified in detail the different steps taken in our application by the different actors, in order to achieve the described action.

1. Audio Dispatcher operator creates a P2P call to a help point, other Audio Dispatcher operator or to an external number.

Actor	Action	Effect
Audio Dispatcher GIS	An Audio Dispatcher operator creates a call to a help point, other Audio Dispatcher operator or to an external number.	If call receiver is an Audio Dispatcher operator the call is presented in Audio Dispatcher console. As an Audio Dispatcher operator on a peer to peer call, he can answer the call, divert (if not conference call) and reject the call. After answering it he can hang up the call, put on hold, put him on mute, transfer the call or add new participants to the call and transform it to a conference call.

2. Create a Conference call.

Actor	Action	Effect
Audio Dispatcher GIS	An Audio Dispatcher operator starts a conference call.	For all Audio Dispatcher participants, the console will present the incoming call. Help points are in auto answer. When the first participant answer, the call is established immediately. When the other participants answered, then they will join the call. If there are Help Points in participants, they will auto answer the call. The Audio Dispatcher operator that initiates the conference call became the call's master and only him has some features to manage the call, such as: • Hold the whole call; • Remove the whole call of hold; • Hold individually the participant; • Remove individually the participants of hold; • Mute individually the participant (mute the participant for the rest of the participants); • Remove the mute individually participants. • Deaf individually the participant (deaf the participant for the rest of the participants); • Remove the deaf individually participants. The command to create a new conference call can come from HMI/GIS system.

3. Emergency button is pressed from help point.

Actor	Action	Effect
Passenger Audio Dispatcher GIS	The passenger or other staff member presses the emergency button to initiate an emergency situation.	When the emergency call arrives in the Audio Dispatcher System, the Audio Dispatcher first searches for the next available operator (with the HOA algorithm). Then, the notified operator will be prompted to accept the call. In the event that we don't answer, the call will be transferred to the next operator. In this case, if the next available operator already has an active emergency call on going, the new call is queued. **HMI** – The incoming call should be presented in the synoptic view. **GIS** - The localization of emergency call is marked in GIS console of the operator.

4. Select Contacts.

Actor	Action	Effect
Audio Dispatcher	The Audio Dispatcher operator select a contact or a set of contacts to create a call (or broadcast) or add participants to a call.	The Audio Dispatcher operator console have a capability to add contacts with the following functionalities: • Select a contact that represents another set of contacts (line, station, etc.); • Select from a list of other contacts; • Select a contact through the autocomplete feature; • Free text number.

5. Divert Call.

Actor	Action	Effect
Audio Dispatcher GIS	An Audio Dispatcher operator transfer a call to other Audio Dispatcher operator, to a specific phone or to a group of phones, before answer it.	The destination of the call can only be one, be it another operator or a central number that represents another sets of numbers. The destination of the call cannot be a HPT or any other device with auto-answer configured. If call receiver is an Audio Dispatcher operator the call is presented in Audio Dispatcher console. Audio Dispatcher receiver can answer, transfer it again or reject the call (if the transferred call is an emergency call, it is not possible reject the call). **GIS** – The localization of diverted call is marked in GIS console.

2.4 Similar Products and Key Differentiators

There are several other products on the market that are capable of managing VoIP communications in a unified manner. Tools such as Skype and Skype for Business, WhatsApp, FaceTime, Google Hangouts and several other proprietary tools from other vendors in IP Telephony market, such as Alcatel Rainbow, to name a few, offer similar or even more complete functionalities than the ones performed by Audio Dispatcher.

However, as the market landscape where Audio Dispatcher is deployed is almost exclusively focused on railway operational control centres, there are specific voice technologies (Radio, TETRA, GSM-R, SIP, analogue, etc), operational rules and interfaces created and adapted in order to bring to a unique voice communication console, all operational voice and text communications between the different stakeholders involved in these complex communication processes.

On top of the different voice technologies integrated, there are a plenty of telephony devices (mobile phones, radio handheld devices, help points, intercoms, standard desk telephony devices, smartphones) and operational rules demanded by railway customers to manage communications based on very specific principles, such as managing call priorities, areas of responsibility management, call location based assignments, call forwarding strategies, emergency group calls, railway emergency calls and several other scenarios that cannot be managed with standard Customer Off The Shelf (COTS) products existing in the market.

Also, high availability scenarios, where passenger safety and security is involved, such as no drop call and geo-redundant architectures, play a role in the specifics of how the Audio Dispatcher software architecture is designed and deployed.

All the previously mentioned arguments are the reason for the existence of Audio Dispatcher. To bring railway operational control centres users a unique HMI front end tool, capable of handling different call management processes, based on specific rules and priorities with an ergonomic, user centric and simple user interface, based on modern web browsers, are its key differentiators.

Chapter 3

State of the Art

This third chapter presents the state-of-the-art of relevant technologies regarding the development of the application. We will start by describing several core components that will integrate the Audio Dispatcher environment, for example, Asterisk and GDP. We will then describe these technologies in more detail in order to better contextualize them in the system architecture that will be presented in Chapter 5.

Next, we will make a brief summary of the voice communication interfaces used in the AD system, which will also be represented in the architecture diagrams presented in Chapter 5.

To finish, we will introduce the remaining tools that we found important to mention, in order to better understand the rest of the content of this thesis where several of these technologies will be mentioned.

3.1 Asterisk PABX Server

Asterisk is a software implementation of a private automatic branch exchange (PABX) that is used to establish and control telephone calls between telecommunication endpoints. Asterisk allows SIP extensions to be registered and provides a programmatic REST API interface that enables the Audio Dispatcher Core component to send commands and receive events from it (e.g., transfer call, make a call, hang-up call, etc.). The SIP Extensions include all devices that allow making or receiving calls, including Help Points, Softphones or SIP desk phones. These components can be either software or hardware based, that register directly in the Asterisk Server and make themselves available to make and receive calls.

3.2 GDP UI Toolkit

GDP (GTS Digital Platform) was created by Thales Group to provide a single unified digital platform with the purpose of incorporating multiple products and services developed by the Ground Transportation System (GTS) worldwide division. It supplies the building blocks for building modern, robust, secure and scalable applications. This platform aims to achieve the following goals: Lower operational cost and easier maintainability, Uniform and intuitive User Interface and shorter project delivery time. In order to help with these goals the GDP team developed the GDP UI Toolkit. This toolkit helps Thales product teams bootstrap their HMIs development, while providing a common integration framework for the front-end of all Thales products.

The GDP UI Toolkit relies heavily on two key technologies: Web Components and EcmaScript (ES) Modules. They will also be using Polymer (and Polymer CLI) to help build Web Components.

For Thales internally, some of the benefits of this Platform are:

- Reduced development efforts;
- Low maintainability costs;
- Centralized optimizations;
- Reuse of common functionalities between applications;
- Homogeneous development methodologies;
- User and Role management;
- Uniform, intuitive, ergonomic UX (user experience).

The AD used up to date technologies and practices, as proposed and supported by the GDP. The HMI was later integrated as widgets in the GDP Dashboard.

The following technical constraints emerged from the development of Audio Dispatcher within the framework of GDP:

- **Cloud Deployment:** Audio Dispatcher 3 must run in a Cloud Foundry installation;
- **Authentication, Authorization and Logging:** Provided by GDP components developed and maintained outside Audio Dispatcher, but operated in the same cloud instance: GDP Keycloak, GDP User Manager, GDP Security Manager, GDP Log Manager, GDP Alarm Manager, etc;
- **User Interface:** Audio Dispatcher 3 uses the GDP HMI Toolkit, which provides a web interface development framework, based on Angular;

- **Messaging:** Inter-service communication in the Audio Dispatcher 3 runtime uses the RabbitMQ message queueing framework;

- **Persistence:** Each Audio Dispatcher 3 service relies on an accessible PostgreSQL instance for data persistence during normal operation.

In the figure below (3.1), we will try to give a better understanding of the potential of this platform and its main advantages in the development and integration of Thales products. In this image is presented a mock-up of the main page of the GDP platform that will be integrated into each operator's system, the central area contains the designs of two different widgets, which are products that will be developed by the Thales development team.

The first widget (left one) called Geo Map, is an application responsible for displaying a map of a city's transport network. In the second widget (right one) we can see one of the widgets to be developed by the Audio Dispatcher (AD) team, denominated Call Manager, responsible with the call management part of our application. Also to notice, that some of the call manager features are integrated by the geo map widget, we can observe this with the call card represented on the left widget.

Integrating the two widgets, although technologically challenging, will enhance the features of both widgets. Thus managing to have a differentiating performance regarding each application domain, while also meeting customer requirements more effectively.

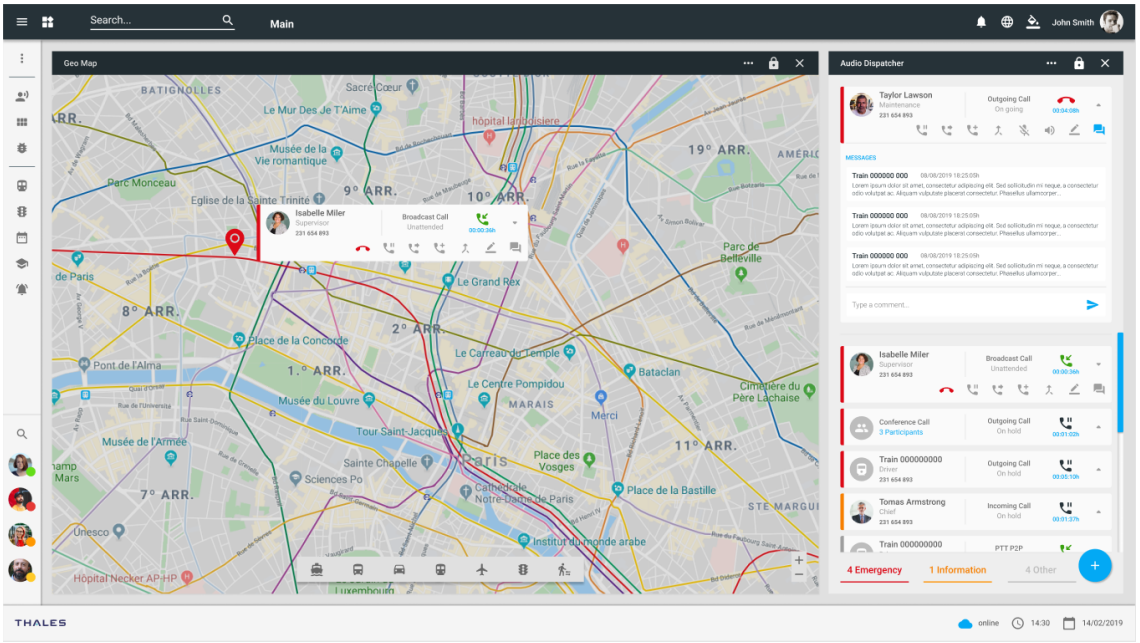


Figure 3.1: GDP Main Page Design

3.3 Polymer [8]

Web components are an incredibly powerful new set of primitives baked into the web platform, and open up a whole new world of possibility when it comes to componentizing front-end code and easily creating powerful, immersive, app-like experiences on the web.

Polymer is a lightweight library built on top of the web standards-based Web Components APIs, and makes it easier to build your very own custom HTML elements. Creating reusable custom elements or using elements built by others can make building complex web applications easier and more efficient.

Among many ways to leverage custom elements, they can be particularly useful for building reusable UI components. Instead of continually re-building a specific navigation bar or button in different frameworks and for different projects, you can define this element once using Polymer, and then reuse it throughout your project or in any future project.

3.4 Voice Communication

3.4.1 GSM-R

The GSM-R interface allows a secure voice and data communication between the Audio Dispatcher system and railway operational staff (Train drivers, station controllers, dispatchers). It is an interoperable track-to-radio technology used by many infrastructure managers and railway undertakings for operational voice communications.

3.4.2 Radio

TETRA (Radio) is a digital trunked mobile radio standard developed to meet the needs of traditional Professional Mobile Radio user organizations such as: Public Safety, Transportation, Government, etc. This interface grants a secure voice and data communication between the Audio Dispatcher system and operational staff deployed in the field.

3.4.3 SIP Trunks

Session Initiation Protocol (SIP) trunking is a service offered by a communications service provider that uses the protocol to supply Voice over IP (VoIP) connectivity between an on-premises phone system and the public switched telephone network (PSTN). Simply put, SIP is the technology that creates, modifies, and terminates sessions with one or more parties in an IP network, whether a two-way call or a multi-party conference call.

The Audio Dispatcher system shall enable the PABX (Asterisk) to interface with SIP Trunks to send and receive calls via internet and PSTNs.

3.4.4 Voice Recording System

The Voice Recording System VRS is responsible for recording all the calls being handled in the system by a dedicated appliance hosted in Audio Dispatcher Cloud environment infrastructure. Call Recording allows clients to search, find and categorize recordings based on date and time of calls, incoming phone number, outgoing phone number or other call record attributes. This component interacts directly with the PABX component (Asterisk).

3.5 Other Tools

3.5.1 Jenkins [10]

The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project. This integration server (Jenkins) detects the committed piece of code and launches the compilation of the program. It runs quality/metric tools onto the source code, runs unit tests and full regression tests onto the committed code, evaluates the coverage of the unit tests over the code, deploys libraries for other teams, and publishes the generated metrics into reporting servers.

3.5.2 SonarQube [20]

SonarQube provides the capability to not only show health of an application but also to highlight issues newly introduced. With a Quality Gate in place, you can fix the leak and therefore improve code quality systematically. Additionally SonarQube supports developers with early security feedback for the code they have written, thereby powering the agile movement in software development. This tool also helps the team by supplying a continuous integration dashboard for automated test status, while also providing static and dynamic code analysis. This analysis is complemented by the usage of a toolkit named JaCoCo that is responsible for measuring and reporting Java code coverage. In Chapter 6 we will explain in more detail how we used this tools in our application code reviews.

3.5.3 Cloud Foundry [4]

Cloud Foundry is a multi-cloud application platform as a service (PaaS) on which developers can build, deploy, run and scale applications. Audio Dispatcher is deployed and run in a Cloud Foundry environment.

3.5.4 Keycloak [11]

To deal with the authentication and authorization of the application, we use Keycloak. Keycloak is an open-source identity and access management solution which mainly aims at applications and services. Users can authenticate with Keycloak rather than individual applications. So, the applications do not have to deal with login forms, authenticating users and storing users.

3.5.5 RabbitMQ [14]

For some Inter-service communications in the Audio Dispatcher, we use the RabbitMQ message queueing framework. RabbitMQ is a message broker software. It accepts messages from producers and delivers them to consumers. It acts like a middleman which can be used to reduce loads and delivery times taken by web application servers.

3.5.6 WebSocket [5]

WebSocket allows a bidirectional and full-duplex TCP socket connection to establish between client and server.

The WebSocket specification defines an API establishing "socket" connections between a web browser and a server. In plain words: There is an persistent connection between the client and the server where both parties can start sending data at any time.

The WebSocket protocol specification defines ws (WebSocket) and wss (WebSocket Secure) as two new URI schemes that are used for unencrypted and encrypted connections, respectively.

3.5.7 PostgreSQL [13]

PostgreSQL is a general-purpose object-relational database management system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. Each AD microservice relies on an accessible PostgreSQL instance for data persistence during normal operation.

3.5.8 Redis [15]

Redis is an in-memory data structure server frequently used as a distributed shared cache because it enables true statelessness for an applications' processes, while reducing duplication of data or requests to external data sources. This tool is used for caching the data exchanges handled by the application.

Chapter 4

Processes and Methodologies

All Thales products adopt Agile methodologies in their development. The Audio Dispatcher project organization includes one Scrum team which can work in parallel on different functional areas. The AD scrum team includes a Product Owner, a Scrum Master, Testers and Developers. Regarding the AD project, our team agreed in holding daily meetings following the guidelines defined by the Scrum method, these meetings had the purpose of analyzing the tasks performed the previous day by each member, while also preparing the tasks to be implemented in the future. We also had biweekly or monthly meetings, where we would review our project's situation in regards to the initial planning, we would then readjust the necessary requirements accordingly.

4.1 Agile

Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly. Whereas the traditional "waterfall" approach has one discipline contribute to the project, then "throw it over the wall" to the next contributor, agile calls for collaborative cross-functional teams.

Open communication, collaboration, adaptation, and trust amongst team members are at the heart of agile. Although the project lead or product owner typically prioritizes the work to be delivered, the team takes the lead on deciding how the work will get done, self-organizing around granular tasks and assignments. Agile is not defined by a set of ceremonies or specific development techniques. Rather, agile is a group of methodologies that demonstrate a commitment to tight feedback cycles and continuous improvement.

4.2 Scrum

Scrum is a process framework which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value. Scrum is not a process, technique, or definitive method. Rather, it is a framework within which we can employ various processes and techniques. Scrum makes clear the relative efficacy of your product management and work techniques so that we can continuously improve the product, the team, and the working environment.

As this generic process description wants to provide guidance as to which components are the vital parts of the implementation of Scrum processes at Thales.

This list comprises the five main pillars that shall be covered when introducing agile on a Thales product (subsystem):

- Aligned sprint length according to enterprise wide delivery tact;
- Standardised Definition of Done (and their product related deviation);
- Provide verification tables as source for verification and validation in a commonly accessible medium put under configuration management;
- Attendant validation and quality assurance should be enabled;
- Continuous Integration.

4.3 Management Tools

To support the followed methodology, it was essential to organise the project's workflow, we used these tools to help with this task.

4.3.1 Jira [2]

Jira is a software designed to help teams manage their work. Originally, Jira was designed as a bug and issue tracker. But today, Jira has evolved into a powerful work management tool for all kinds of use cases, from requirements and test case management to agile software development. This is the tool used by Thales that helps the AD team with project management by following these actions:

- Capture and organize your team's issues (bugs or tasks);
- Prioritize and take action on what is important;
- Monitor the status and evolution of the product.

4.3.2 DOORS [9]

IBM Rational DOORS is a requirements management application produced by IBM for optimizing requirements communication, collaboration and verification throughout your organization and supply chain. This scalable solution can help you meet business goals by managing project scope and cost. Rational DOORS lets you capture, trace, analyse and manage changes to information while maintaining compliance to regulations and standards. This tool helps with the development and management of the product's requirements while also supporting the management of test procedures and then link them with their appropriate requirement.

4.3.3 Git [7]

Git is a distributed revision control and source code management (SCM) system with data integrity, and support for distributed, non-linear workflows. The tool Git in itself is the core of both client and server tools, which themselves add a layer of presentation enabling to manipulate the tool in a easy manner. These were the main functions of our Git usage:

- Repatriate source code and documents;
- Save changes in history;
- Create different releases versions;
- Merging workflow management;
- Browse history.

4.3.4 Gerrit [6]

This is the tool used to help with the code review of the Audio Dispatcher. Gerrit is a web-based tool that is used for code review. Its main features are the side-by-side difference viewing and inline commenting, this can be really useful, when reviewing other developers modifications. Code Reviews are done continuously and must answer the following questions:

- What changed?
- Are the changes functionally correct?
- Are the changes in accordance with the coding practices for the solution?

If the answers to the questions are satisfactory, a positive review is given. In that case, comments can also be added (as general comments or over specific parts of the changes). If the answers are not satisfactory, then a negative review is given. In such case, of a negative review, a comment shall be added explaining what is wrong.

Chapter 5

Project / Approach

In this chapter, we will start by reviewing the solution architecture and its main elements. Following the architecture, is presented the two data models used for the database management of the application.

Afterwards, will be a section focused on the Frontend part of our application. This section will be divided into three parts, the three main widgets, **Call Manager (compact)**, **Call Manager (expanded)** and **Contact Manager**. We will start by presenting multiple designs, and the decisions behind each widget main features. In the end of each part, will be displayed some screenshots of the end product of our application, where you will be able to see the outcome of my work.

5.1 Solution Architecture

This section describes the high-level architecture and the Audio Dispatcher core components. Figure 5.1 shows the high-level view of the system's physical architecture where the product is integrated. As the Audio Dispatcher is built on microservices, our solution is designed as an Event-Driven Architecture.

The Audio Dispatcher Server runs within the Cloud Foundry environment, which is the cloud application platform used to deploy the AD application. This platform accesses the PostgreSQL Database in order to store the required information. The server will interface with a Network Time Protocol(NTP) server which is a protocol used to synchronize clock times between computer systems in a network. The AD server also connects with an LDAP server used as the authentication entity that also stores users account details.

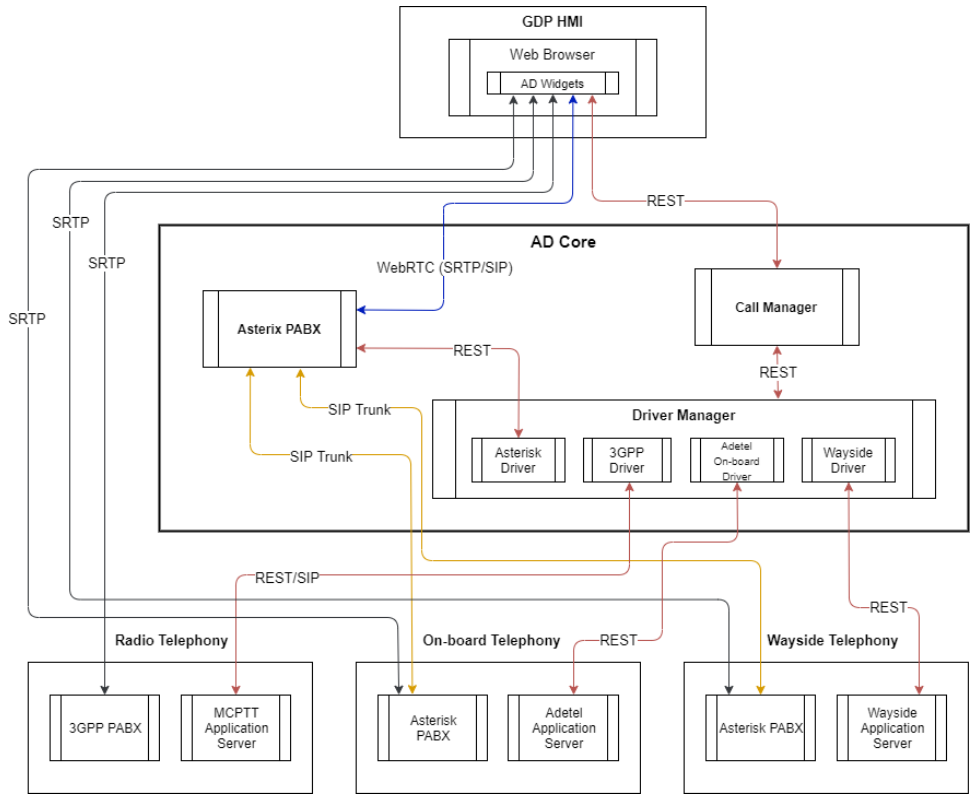


Figure 5.1: High Level Architecture

Another core component of the Audio Dispatcher system represented in Figure 5.1 is the Asterisk Server. Asterisk is a software-based telephony server that allows Session Initiation Protocol (SIP) extensions to be registered, e.g, Help Points, train driver consoles, control centres. It also provides a software API that enables the Audio Dispatcher Core component to communicate with this PABX (Asterisk). Typically, this component registers the SIP extensions of the Help Points and the SIP extensions of the operators in the railway control centres. It will also interface with SIP trunks from any external PABX servers. In order to establish calls to external numbers, for instance emergency services, additional SIP trunks can be configured for the 3^{rd} party PABX servers that will interface with the outside world.

The User interface of the AD application is represented as the AD Widgets component and as Figure 5.1 shows, it is integrated on the web interface framework developed by Thales Group, named GDP UI Toolkit (represented in the Figure 5.1 as GDP HMI). This toolkit relies heavily on two key technologies: Web Components and JavaScript Modules.

Web components are a set of web platform APIs that allow you to create new custom, reusable, encapsulated HTML tags to use in web pages and web apps. Custom components and widgets build on the Web Component standards, will work across modern browsers, and can be used with any JavaScript library or framework that works with HTML, as our web components were developed using HTML, CSS, and JavaScript.

The toolkit uses the Polymer 2 library for building Web Components. Polymer is a JavaScript library created by Google that lets you build encapsulated, reusable Web Components that work just like standard HTML elements, to use in building web applications.

Audio Dispatcher also relies on third party software tools responsible to support its operation and deployment, e.g., (RabbitMQ [14], Redis [15], PostgreSQL [13], these tools were described in detail on the State of Art (Chapter 3).

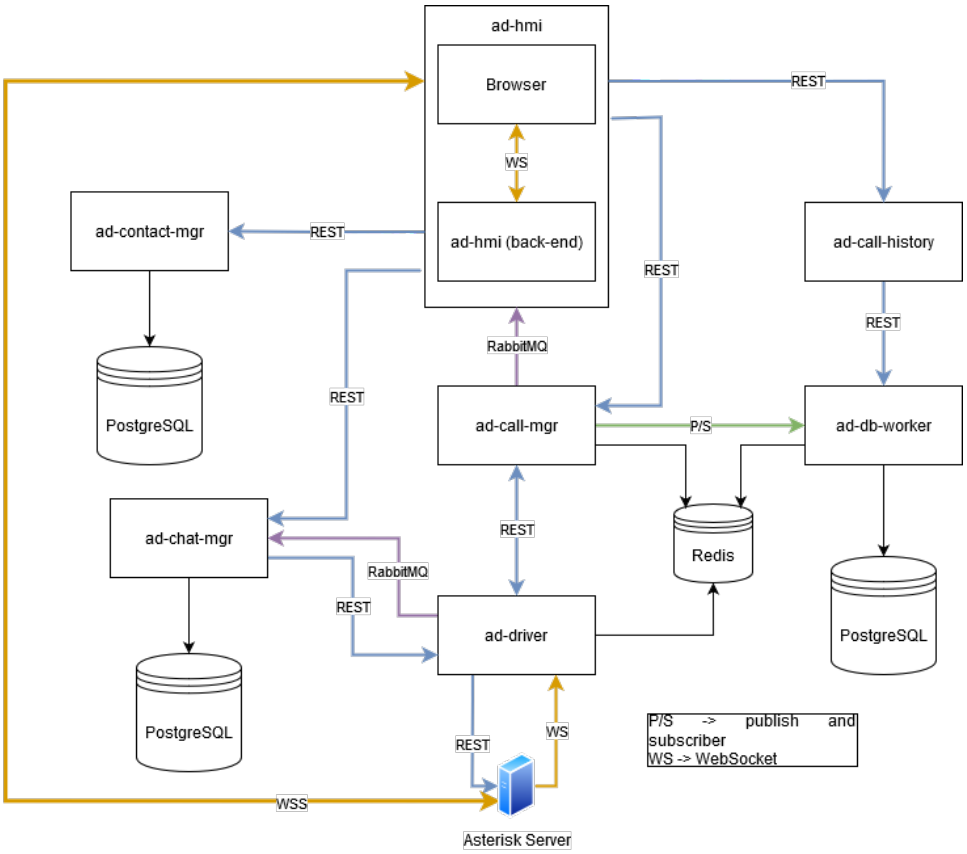


Figure 5.2: Audio Dispatcher Technical Design

This image (Fig. 5.2) displays in a more detailed technical way what the Audio Dispatcher (AD) product is and how it is structured. All Audio Dispatcher modules implement the same administration interface, and therefore are being managed in the same manner. The employed technologies that ensure this consistency are Spring Boot Admin and Spring Boot Actuator endpoints.

This image also illustrates how the several services communicate. In the section below (Middleware 5.2), we will explain these communications in detail.

5.2 Middleware

Communication with the user is performed by means of browser-based user interface. The Frontend of Audio Dispatcher runs on the user machine, while the Backend runs on dedicated, remote environments, which may be deployed either in cloud or on actual hardware. Communication from the browser UI towards the backend services uses REST interfaces. Communication from the back-end services towards the browser UI uses Web Sockets. Service requests exchanged between Audio Dispatcher backend modules uses REST interfaces.

The methods by which the components communicate with each other can differ between Rest endpoints, Websockets or RabbitMQ queues.

In the subsection below, we will explain how the communication between modules is performed using events.

5.2.1 Events

In this section, we will describe how the communication works, using events, between these three modules (ad-call-mgr, ad-hmi (back-end) and the Browser). These were the modules I worked on.

5.2.1.1 Call Manager –> HMI (Back-end)

As seen before (Fig. 5.2), the communication between the **ad-call-mgr** and the **ad-hmi (back-end)** is done through queues (RabbitMQ).

In these queues the object that is forwarded is a *CallManagerEvent* object. This object is composed of the following properties:

- *hmiNotificationType* – indicates the action that was performed by the user and what event should the browser receive from the HMI (ad-hmi (back-end));
- *Call* – the Call object.
- *participants* - a collection containing objects of type *CallManagerEventCallParticipant*.

  The object *CallManagerEventCallParticipant* has these properties:

  - extension – the user's extension;
  - channelId – the user's channel Id;
  - EventType – the event to send to this user (described in the subsection below (5.2.1.2).

The events transferred to each user will depend on the action performed and the person who performed. Depending on the call's mode, Peer to Peer (P2P) or Conference, there are different events.

5.2.1.2 HMI (Back-end) -> Browser

The **ad-hmi (back-end)** upon receiving the notification from the **ad-call-mgr**, it only needs to create an *EventCall* object based on the *CallManagerEvent* object received and send it to the respectively browser. Depending on the event type received, the user interface will adjust the display accordingly.

For example, when a user (Browser 1) makes a call to another user (Browser 2) the payloads (*EventCall* object) sent to each browser are:

- **Browser 1 (caller)**:

Atribute	Value
HmiNotificationType	P2P_CALL
EventType	P2P_WAITING_ANSWER_EVENT

- **Browser 2 (callee)**:

Atribute	Value
HmiNotificationType	P2P_CALL
EventType	P2P_INCOMING_INFORMATION_CALL_EVENT

When receiving these payloads, the interface for each browser will change depending on the kind of message. In these examples, the browser 1 changes to only display the option of cancelling the waiting call. While the browser 2 will present the user with two choices, Answer call or Reject call, to answer or reject the incoming call, respectively.

This communication is performed through Websockets. WebSockets is an event-driven API, when messages are received, a message event is sent to the WebSocket object.

The browser will connect to a specific websocket using the user's associated extension. There are currently 3 websockets connections in use:

- The first one is used to receive notifications about call events.

```
/websocket/user/<username>/call/queue
```

- This one is used to receive the status of each operator.

```
/websocket/broadcast/operator/status
```

- The last one is used to receive notifications about chat events.

```
/websocket/user/<username>/chat/queue
```

5.3 Database

The database was developed and planned by the AD team before I joined them. In this section we will describe the two Database used in the system, namely, the Calls' History data model, and the Contacts data model.

5.3.1 Calls Data Model

5.3.1.1 Redis

Every time a call is made or when any of its interactions (ringing, answer, transfer, etc) are performed, they will be stored on a Redis in-memory database. There is also a mechanism to save this calls from Redis to a more permanent PostgreSQL database in order to store the calls' history.

Each Redis database entry will be composed of a string (bridge ID of the call), as the key, and a Call (Object), as the value.

5.3.1.2 PostgreSQL

In the more permanent database, where the calls' history is stored, the structure was followed as described in the picture below (Fig. 5.3):

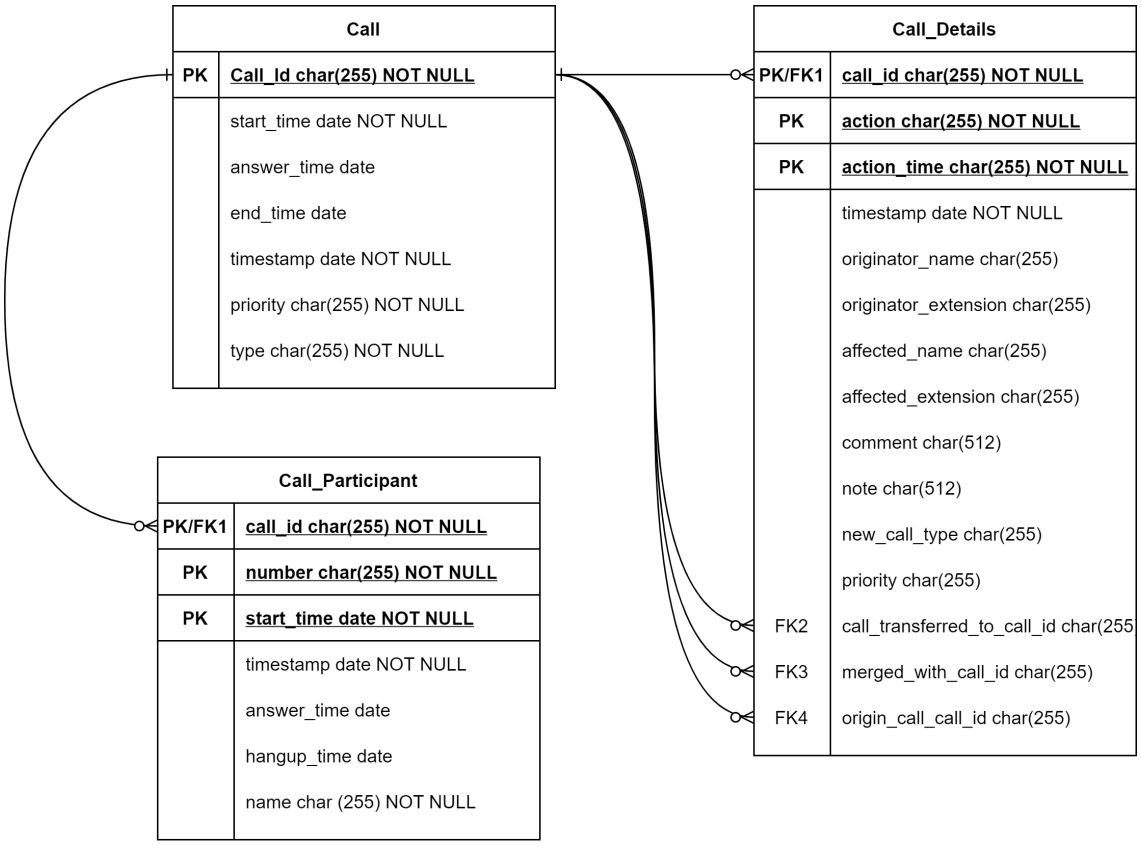


Figure 5.3: Audio Dispatcher Calls Database

The table **Call_Details** will be filled with the different actions performed during a call. Below (Fig. 5.4) is an example of several actions made on a call.

action	action_time	call_id	affected_extension	affected_name	comment	new_cal_type	originator_extension	originator_name	priority	timestamp
MAKE_CALL	2021-03-18 17:01:39.477	00ea0f27-6cdb-4a14-9ea6-caf3d29c2c30					716	Nuno		2021-03-18 17:01:40.427
ANSWER	2021-03-18 17:01:46.504	00ea0f27-6cdb-4a14-9ea6-caf3d29c2c30	706	Diogo			706	Dino		2021-03-18 17:01:46.621
ANSWER	2021-03-18 17:02:08.59	00ea0f27-6cdb-4a14-9ea6-caf3d29c2c30	703	João			703	João		2021-03-18 17:02:08.687
HOLD	2021-03-18 17:03:03.734	00ea0f27-6cdb-4a14-9ea6-caf3d29c2c30	ALL	ALL			716	Nuno		2021-03-18 17:03:03.734
HANGUP_PARTICIPANT	2021-03-18 17:03:35.011	00ea0f27-6cdb-4a14-9ea6-caf3d29c2c30	706	Diogo			706	Diogo		2021-03-18 17:03:35.07
CALL_TYPE	2021-03-18 17:03:35.011	00ea0f27-6cdb-4a14-9ea6-caf3d29c2c30	706	Diogo		P2P	706	Diogo		2021-03-18 17:03:35.074
HANGUP	2021-03-18 17:04:00.906	00ea0f27-6cdb-4a14-9ea6-caf3d29c2c30	ALL	ALL			716	Nuno		2021-03-18 17:04:01.009

Figure 5.4: Audio Dispatcher Call Details Table

5.3.2 Contacts Data Model

After a brief description of this model's entities, we will present the Entity Relationship Diagram of the Contacts PostgreSQL Database (Fig. 5.5). In this image we will be able to see how the users, contacts and filters are connected. It will also show how the data of each entity is represented.

5.3.2.1 Entities

- **User**

  A user is someone who is registered in the GDP User Manager application which in turn can login in the AD application.

- **Contact**

  A contact is an entry which the user interacts with to establish a call. At the moment there are three types of contacts:

  - *Simple* - A contact with an assigned extension. This extension will not change. E.g.: trains, help points, firefighters, police, etc.
  - *AD User Operator (AD_USER)* - A contact that has associated a user from the GDP User Manager. Only the operators that work with the AD interface directly will have this type of contact.
  - *Group* - A static group of contacts, which might contain contacts of various types, including other groups.

  One important notion to have with this entity is that, a user is a contact but a contact may not be a user.

  This statement means that an Operator (for example, that is in the command center) will be able to login and will have an extension associated, so will be both a user and a contact. On the other hand, a Help Point will be exclusively a contact and will not have a GDP user associated.

- **Filter**

  A filter is a combination of contacts that one user can see in the interface.

  Each AD Operator (contact with type AD_USER) can have up to 14 different filters including a "Favorites" filter. The "Favorites" filter can only house up to 6 contacts, while the other 13 filters can have as much contacts as the operator wants. These filters limits are a result of the restricted space in the compact version of the Call Manager.

  A filter can be marked as shared, if a filter is to be shared with every operator in the same control center for example.

5.3.2.2 PostgreSQL

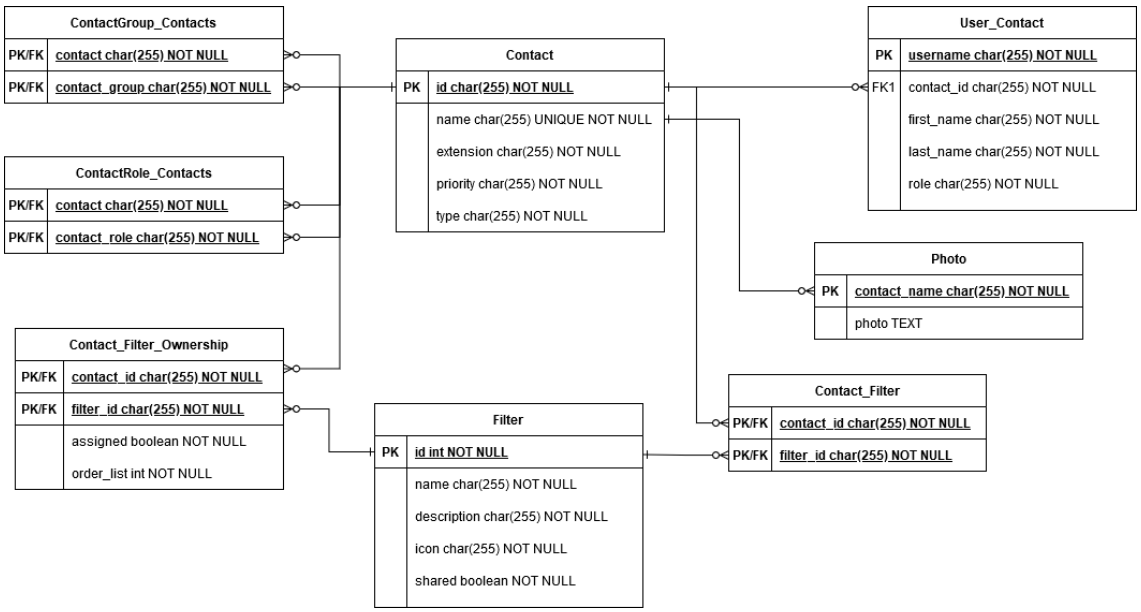


Figure 5.5: Audio Dispatcher Contact Database

5.4 FrontEnd

When we began working on the FrontEnd of the product, we already had multiple meetings discussing the project, and by this time we had a good idea of what we wanted the application to look like. It was very gratifying to have the privilege of matching the beginning of my collaboration at Thales with the start of the FrontEnd of the AD application, which led me to be the main developer of this component.

In the beginning, I immediately faced several challenges, among which I can highlight the fact that the GDP team chose the Polymer technology for the web development of their platform. This technology seemed to be very promising, however, very early on, google decided to abandon this project. This led to several problems during the development of the application such as the lack of online information due to the low adoption of this technology and the little information that existed was outdated. Several situations like this made it impossible to make the best use of several of the Polymer's functionalities.

Another big challenge we faced was the fact that our application was planned and designed depending on external circumstances. One of those circumstances was the fact that the AD widgets were being developed to be integrated into the GDP platform.

Since GDP was still at an early stage when the AD product was integrated, the GDP platform still showed several weaknesses. This caused our team to encounter several problems, where the GDP team failed with several dates and promises previously discussed. These failed promises meant that our team could not benefit from the GDP's platform's advantages, leading to minor delays in our application's development.

One of the main advantages of this platform was the easy integration and development of applications and web components, but this did not happen, since several components planned to be used in the AD project were not developed yet, such as, cards, buttons, tabs, search bar, etc. Which meant that I had to develop them from scratch, not experiencing the advantage of reusing components between the different applications integrated into the GDP platform. This could become a problem because in the future there might be inconsistencies between the different Thales applications, which could lead to unnecessary code refraction.

After these minor setbacks, we settled on developing three HMI widgets. The first is the Call Manager (compact version). This widget would end up being the primary view of the application, where the users would be able to perform most actions regarding the call management part. The next one is the Call Manager (Expanded version). As the name indicates this is the expanded view of the compact version. This view will give users more options when using the application. The contact manager ended up being the last widget developed. This widget purpose is the contact management part, where the users will be able to organize their contact list and their contact groups.

In order to have a clearer idea of the development of the Frontend, this section will be divided into three segments, the three widgets mentioned above, and each of those segments might be separated by versions.

5.4.1 Call Manager (Compact Version)

5.4.1.1 Prototype - Call Manager (Compact Version)

In the first meetings, we began discussing alongside the design team the mockups of the first prototype of the Call Manager widget, and after a few sessions, we started to work on the Audio Dispatcher Frontend.

Below are the initial designs of the first prototype, following the designs will be a brief summary of the thought process behind the decisions made for each feature and its components.

On this prototype (Fig. 5.6), we chose to go with this style and size for the main widget (Call Manager) in view of the fact that our application is planned to be integrated in a unified platform with other applications, so it was required we build a compact version of our product in order to be able to fit alongside the other applications in the same web interface.

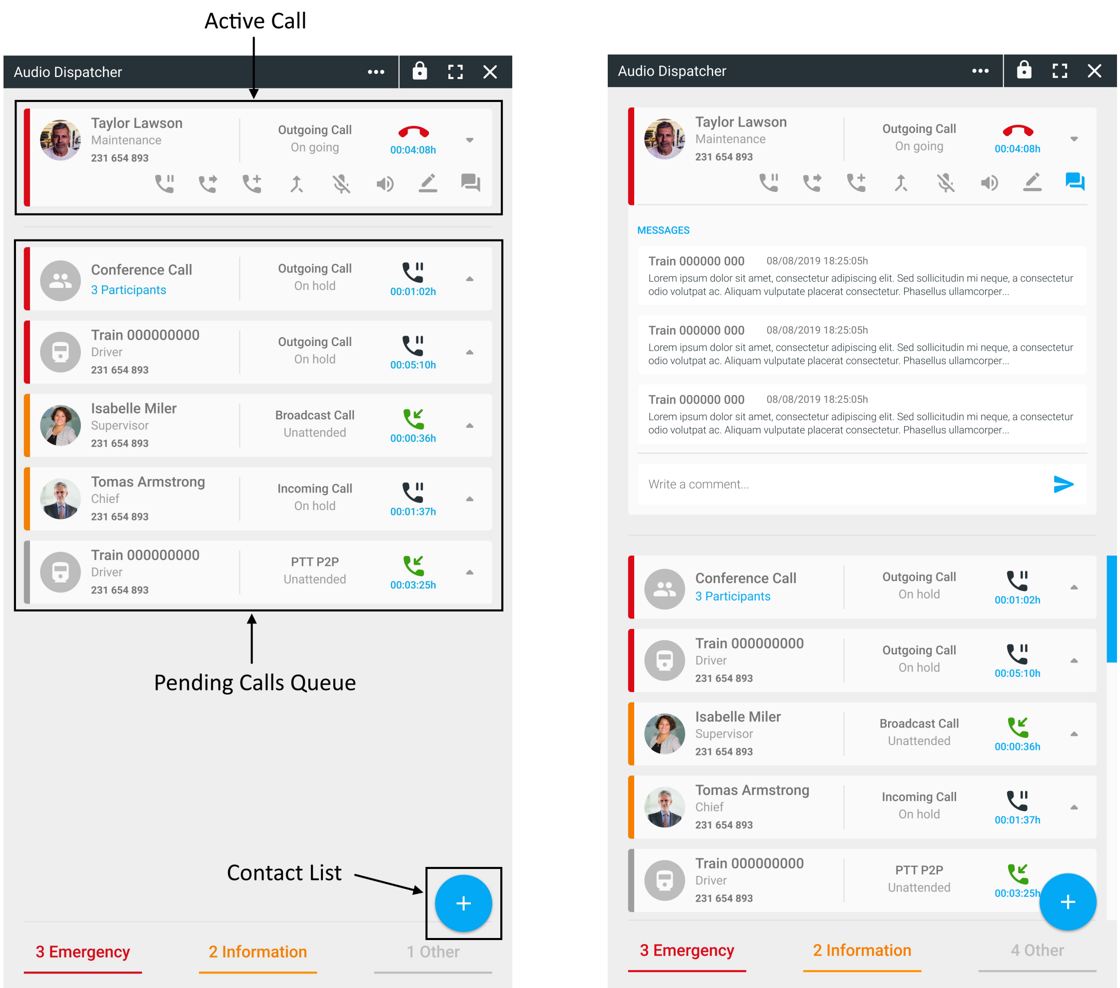


Figure 5.6: First Prototype - Call Manager (Compact Version)

Here we knew we wanted to go for three main elements for this widget, the active call, the pending calls queue, and the contact list.

The active call, where we can see the current call in progress, in this area it was important we could fit the most information possible without everything being too packed. Below the active call, we can find the pending calls area, this is where all the pending calls the user is a part of, either as a participant or, for example, an emergency call being made in its area of responsibility.

For the active call and pending calls, we opt for singular cards for each call as we can see in figure 5.7, in this card we wanted to fit a large amount of information, such as, call timers, call priority, call status, and participants info. We also needed to fit the call functionalities buttons, namely, hold call, transfer call, add participant to the call, merge calls, mute/unmute mic, call volume and call messages, even though some of these buttons would be purely aesthetic. On the pending calls cards, we chose to hide the functionalities buttons in a drop-down section to fit more calls into the area.

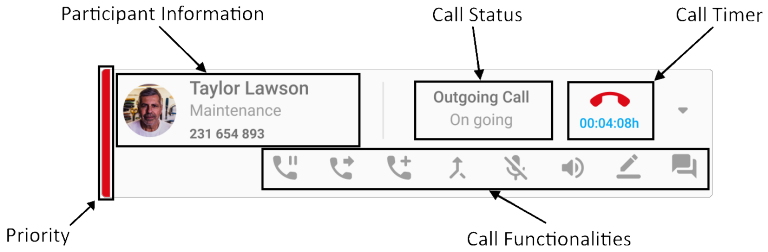


Figure 5.7: Active Call Card

As we can see in the images of the prototype (Fig. 5.6), right away, there were some problems, for instance, the hidden contact list. At this stage, we were still trying to figure out the best way to include the contact list on the application. We concluded that we had two options, we could have a different widget with the contact list, or we could fit a small contact list area on the call manager widget to shortcut all the call operations on the same widget. We opt to go for both approaches to give the user the liberty to best use the application as he preferred. Another issue with this prototype was the confusion with the different areas, as a result of this, the user would have some problems distinguishing the active call area with the pending calls area. With this in mind, we needed to change this as we prepared the first official version of the Audio Dispatcher.

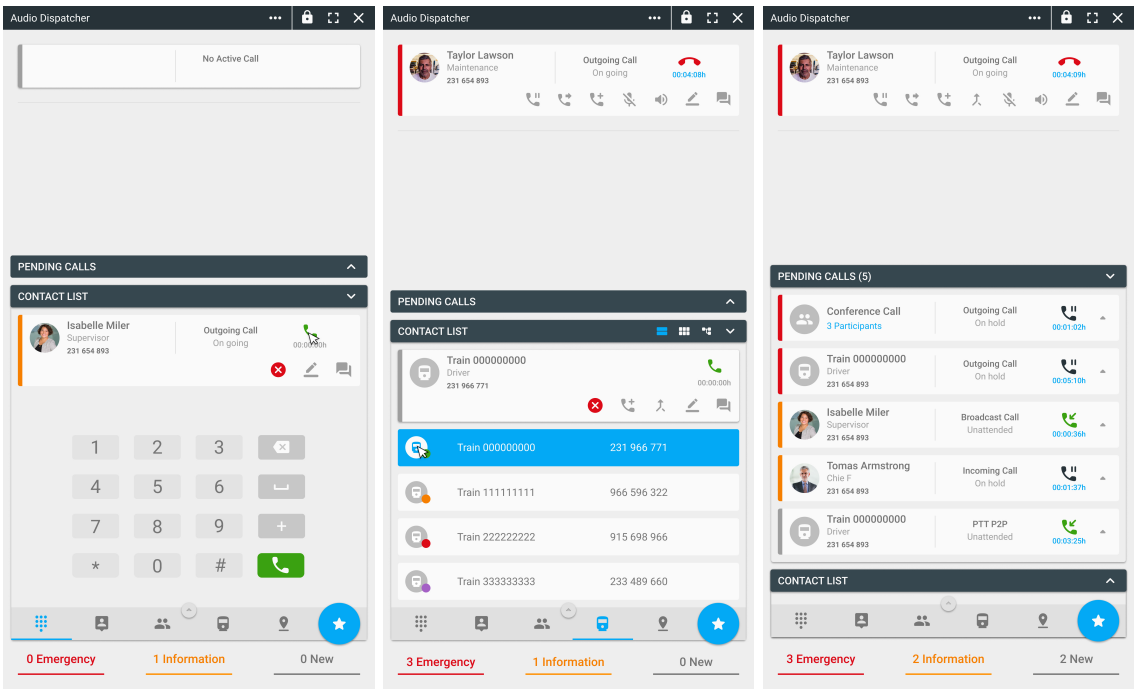
5.4.1.2 First Version - Call Manager (Compact Version)

After working on the prototype for a period of time, we started developing the first version. In this version we had to solve the problems encountered in the development of the prototype while also facing new challenges. In this version we polished the main features as we were getting ready for the first release.

This release would highlight the following features:

- Active Call area;
- Pending Calls area;
- Contact List area with customizable filter tabs (Favorites, custom groups);
- Contacts with User Status (available, busy, offline, etc);
- Dial Pad;
- Management of Peer-to-Peer Calls:
  - Make call;
  - Answer call;
  - Hang up call;
  - Hold/Unhold call;
  - Mute/Unmute call.

And this is what the first version of the call manager looked like:



1 2 3

Figure 5.8: First Version - Call Manager (Compact Version)

As we can see, in this version we decided to make the contact list visible, unlike the initial prototype where it would be hidden in the floating button on the bottom right corner. We decided to do it this way because we thought it would be much more intuitive for the user to have an area that could shrink but still appear on screen, this same technique was passed to the pending calls, this made the user have more control over the application and its components. The floating button was changed to be the user's favourites. These were the most significant changes when we started programming this widget.

5.4.1.3 Final Version - Call Manager (Compact Version)

In the last stage of this widget, we added conference calls and introduced the idea of a call master/moderator, which would, as the name suggests, control the call and its participants with new call functionalities. With this new types of calls we had to redefine the active call card, as we can see in the image below.

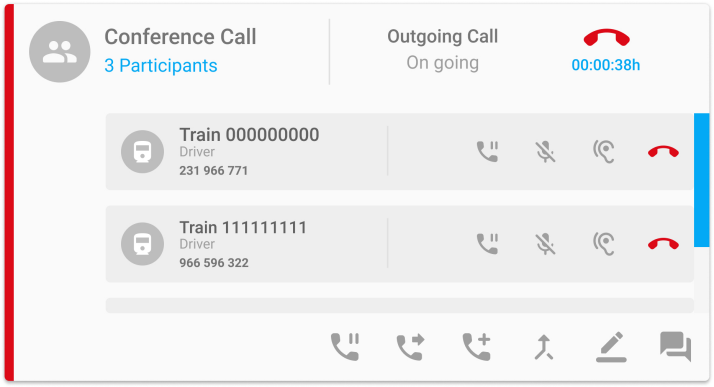


Figure 5.9: Conference Call Card

After this important step, we could focus on the last required features for call management already implemented in the back end, which revolved around the communication between multiple AD operators, such as, transfer call, divert call and merge calls.

The transfer call feature, is the ability to bump a call from one user to another. Based on our requirements, we needed two different approaches, a blind call transfer and an attended call transfer. The first one is a transfer to another extension without actually initiating a call to the end destination. The call is blindly transferred to the destination. An attended transfer is a transfer where before actually transferring to the destination, the call is put on hold and another call is initiated to confirm whether the end destination is able to take the call or not.

Next, is the divert feature, the idea behind this one if treating it as if we wanted to blindly transfer a pending call to another operator, so if we wanted to divert a call we needed to choose the call, then select the end destination, and the call would end up in the pending calls queue on the selected operator, this feature might come in hand if the operators find themselves with too many pending calls.

The last important requirement for this version was the possibility to merge multiple calls. In this feature, we took a little more time to find the most intuitive way for the user to conclude this operation.

We decided that it was necessary for the user to be a master of an active conference call or a participant of an active P2P call, after this necessity was met the user could choose a pending call and finish the process of merging both calls. It ended up as a three-click operation, but we were happy with the final result.

This release would highlight the following features:

- Management of Conference/Group Calls:
  - Add participant;
  - Remove participant;
  - Individually mute/unmute participant;
  - Individually hold/unhold participant;
  - Individually deafen/undeafen participant;
- Transfer Call;
- Divert Call
- Merge Calls;

And this is the result of the final version of the Call Manager (compact version):

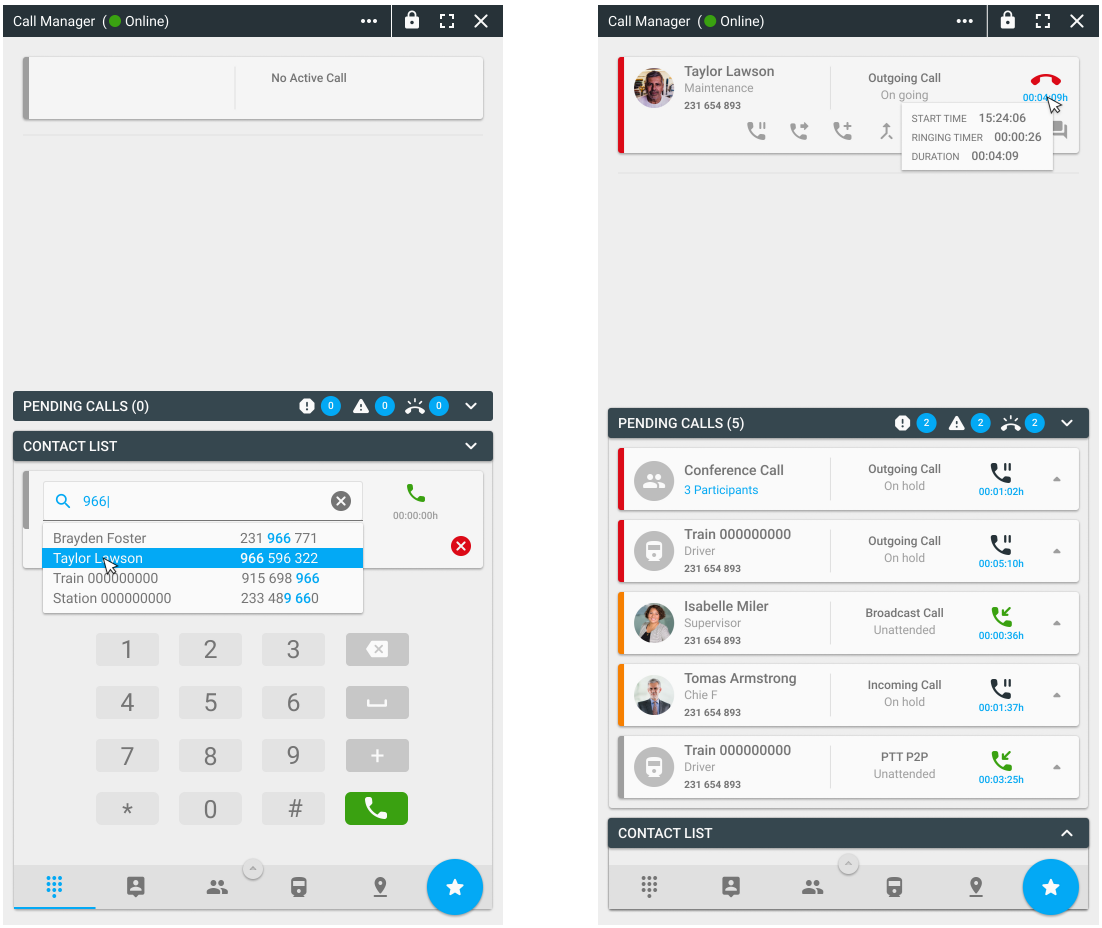


Figure 5.10: Audio Dispatcher Call Manager Final Version - 1

In these images, we can see the features of the *search bar* (left image), and *call timers* (top right corner of the image on the right). These were important requirements for the product and the clients since they are important features in the context of call applications.

We also reviewed some design choices and made some changes, for example, the zone below the contact list area that was meant for filtering the pending calls. We decided to change it to the header of the pending calls where it made more sense and gave us more space to work with.

Finally, we can also notice the green circle on the widget's header next to the name Call Manager, where the user will be able to check their connection to the server. This is an important feature because if there are connection problems while in calls, the user will be able to discover whether those problems are coming from him or the destination.

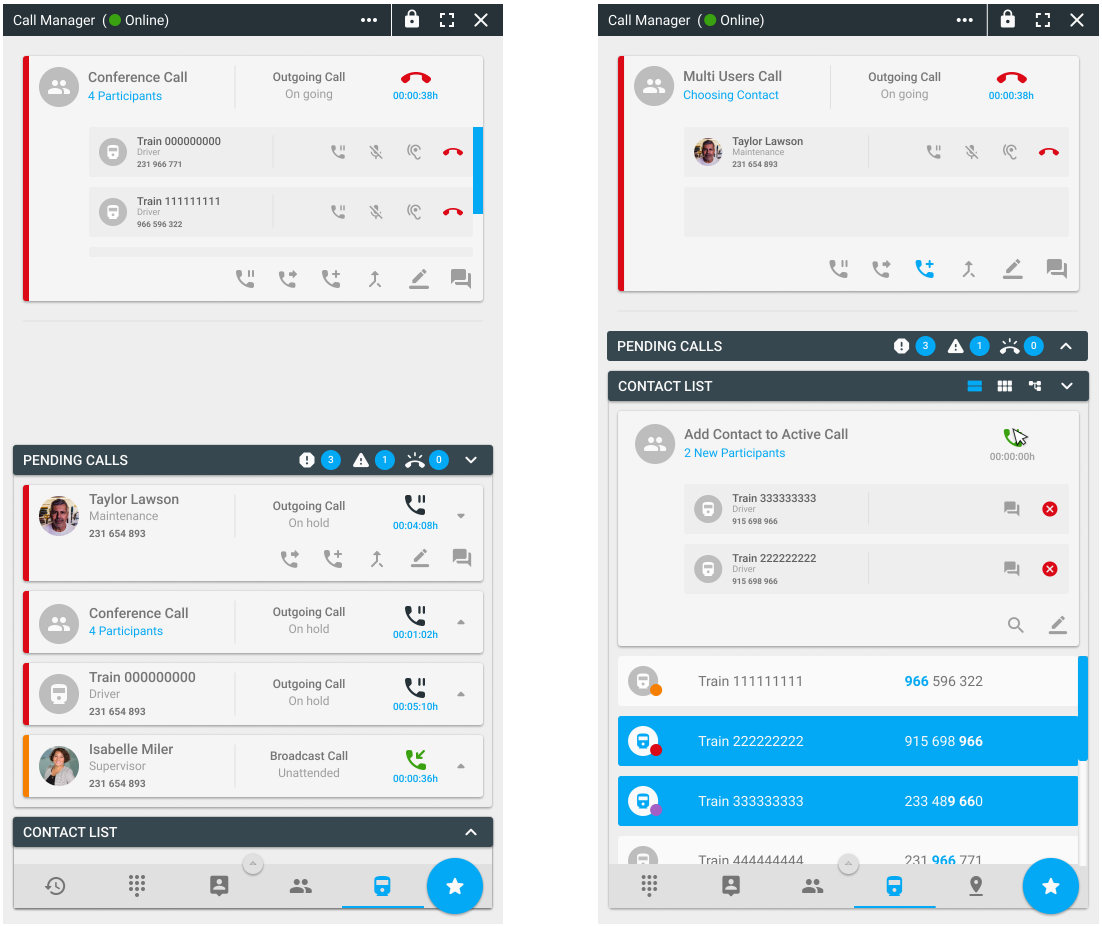


Figure 5.11: Audio Dispatcher Call Manager Final Version - 2

On these next images, we can see basic operations on conference calls. On the left, we can observe a conference call as an active call, this was the most important feature to add to this version since it gave us the possibility to develop the main call operations of a control center.

In the next image, we can notice the method where we decided to add one or more participants to a call. In this case, we would be in a P2P call and we would change to a conference call when we completed this operation. Note that the new master of the conference call would be the operator that added the participants to the call, this is important to note because several conference call operations in our application are only available to the call master, such as muting all participants or removing participants from the call.

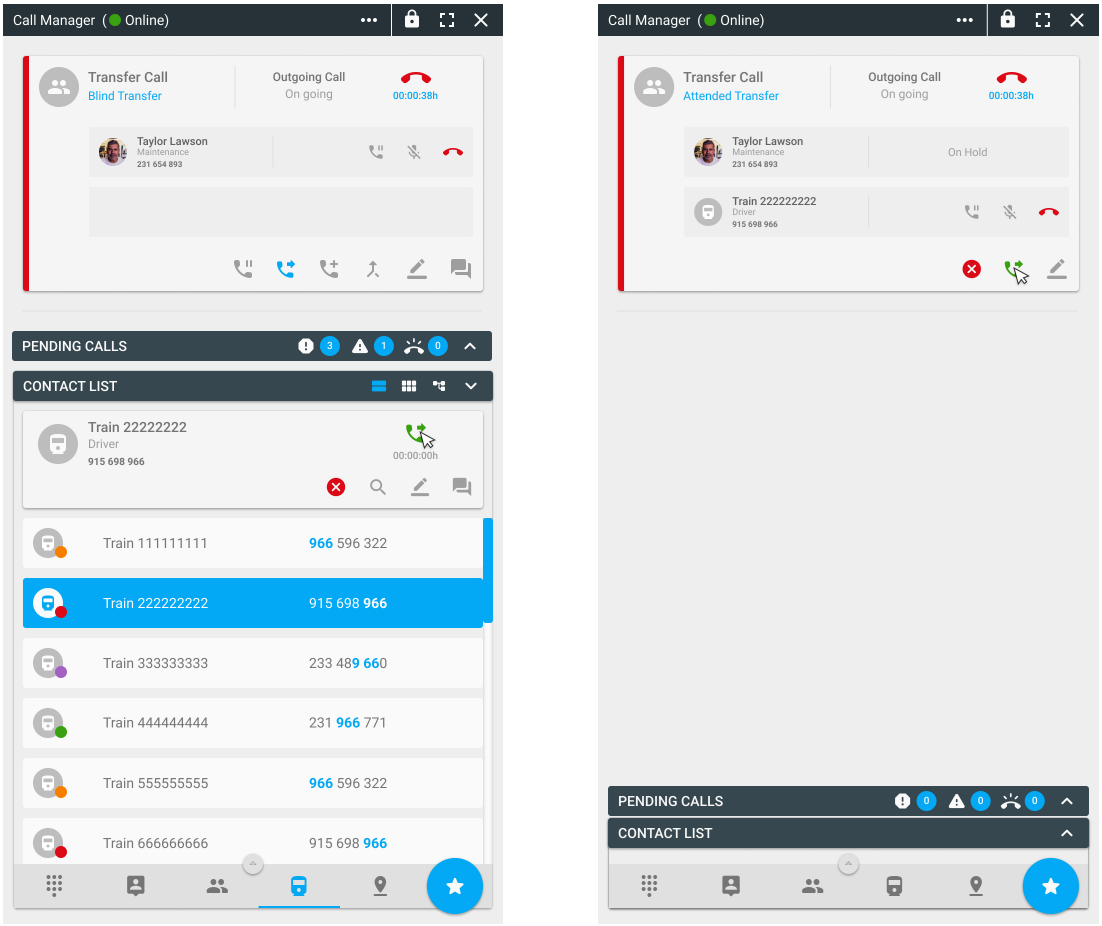


Figure 5.12: Audio Dispatcher Call Manager Final Version - 3

Here, we can see the operations *Blind transfer* (left image) and *Attended transfer* (right image). In the first we can see that the operator is always on call with the person that is going to be transferred, however, he doesn't communicate with the destination that he is going to transfer the call, he just concludes this operation while only informing the participant on the call. This method managed to be a 3-click process which is very quick and intuitive for the user.

In the second image, we can observe an example of an attended transfer where the operator is already in the last step of this operation. Notice that the participant to be transferred is put on hold while the operator communicates to the destination if it is possible to complete the transfer. This is one of the most used operations in the context of control centers, so we had a little more consideration when developing it. In the end, it was not as fast or intuitive as the blind transfer but considering the complexity of this operation, our team was happy with the final result.

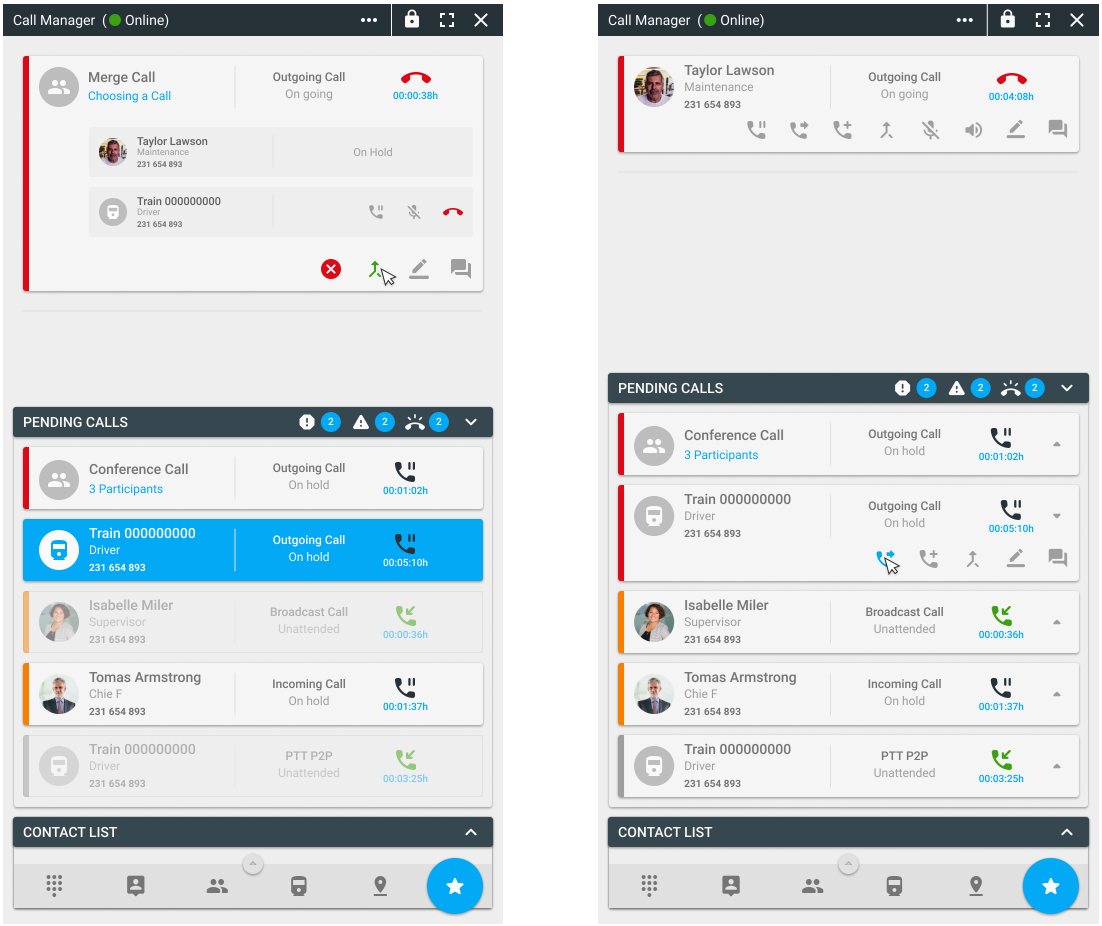


Figure 5.13: Audio Dispatcher Call Manager Final Version - 4

In these last images, we can see the remaining operations added in this version. *Merging calls* (left image) and *Divert a call* (right image). In the first image, we can see a user in a P2P active call in the middle of the merging operation of two P2P calls in which he is a participant. As we can observe, he has selected a P2P call from his pending calls list, represented with the blue background on the pending call card. Note that the operator can only merge calls placed on hold, as we can see by the unanswered calls in his pending calls list that become disabled when the operator starts the merging call process.

The last image shows the operator starting the divert call operation. This operation is available on all pending calls cards in the menu hidden by the arrow on the right side. The operator after starting this process selects the contact he wants the call to be transferred to and then completes this operation. Note that you do not need to inform any participant that you are going to divert the call, just transfer the call from your pending call list to the pending calls list of the new operator.

5.4.2 Call Manager (Expanded Version)

As the product progressed, the development team was presented with more requirements for the next release, but we faced a new challenge, we needed more space, as we could not fit all the new features on the compact version. So we decided to take advantage of the GDP interface features, namely the maximize widget one. In order to use the expanded version new features, the user would need to maximize the Call Manager widget window which meant he couldn't use the other applications at the same time.

5.4.2.1 First Version - Call Manager (Expanded Version)

First we needed to rebuild the application in order to be able to integrate an expanded version, this was more complicated that it appeared and was delayed for some time. As we can see in the figure 5.14, we decided to relocate the pending calls queue to the left side so everything would end up with a reasonable size.

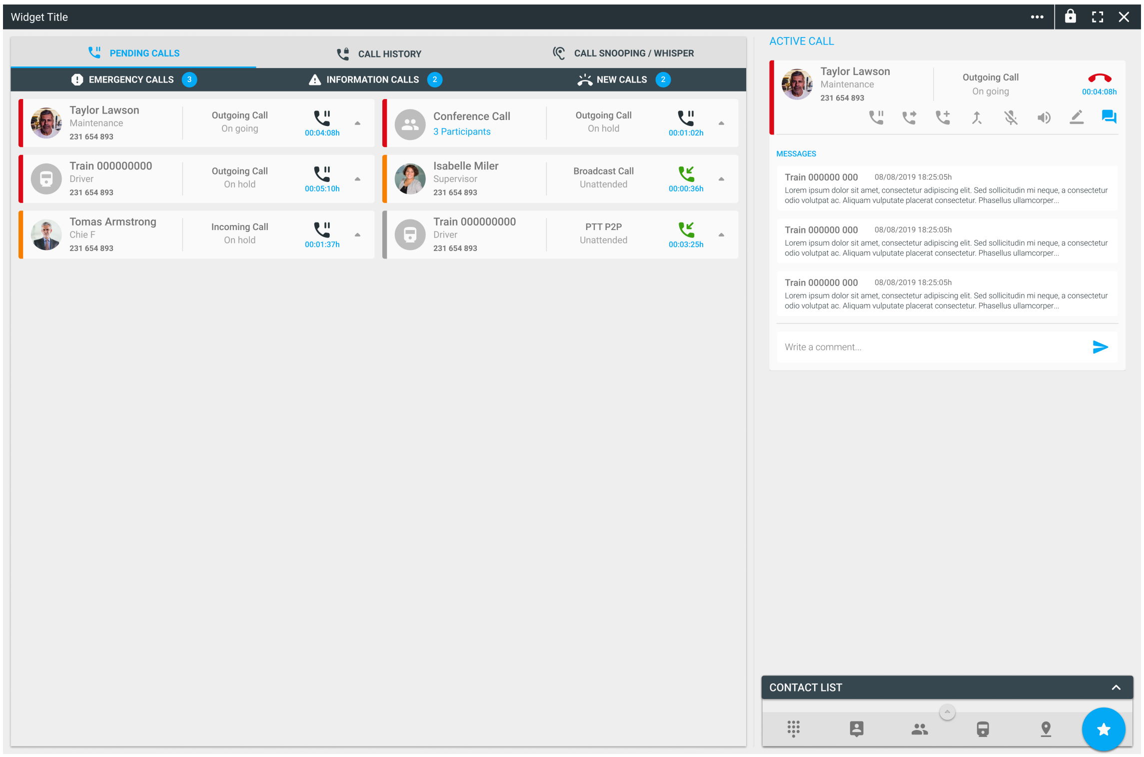


Figure 5.14: Call Manager (Expanded Version) - Main Page

After we were capable of unifying both versions (compact and expanded), we were ready to work on the new features, the main feature of this release was the Call History, this was an essential component of our application as the operators would be able to analyse the calls' history in the event of an accident or another incident to see if how, when, and why something went wrong.

The Call History ended up looking like this:

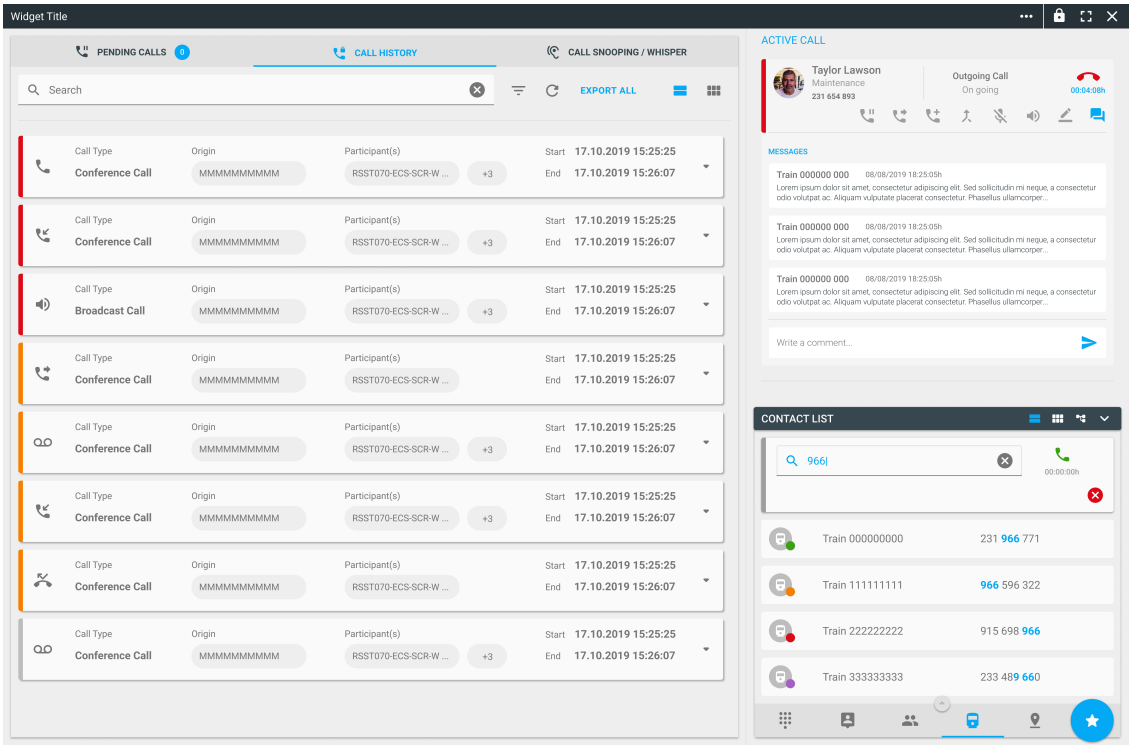


Figure 5.15: Call History - Main page

For the Call History (Fig. 5.15), we opted to go for the same card styles as the rest of the areas, but because we had too much information for each call, we needed to decide what were the primary details to show on the main page, while the rest of the information would be hidden in a drop down similar to the pending calls one.

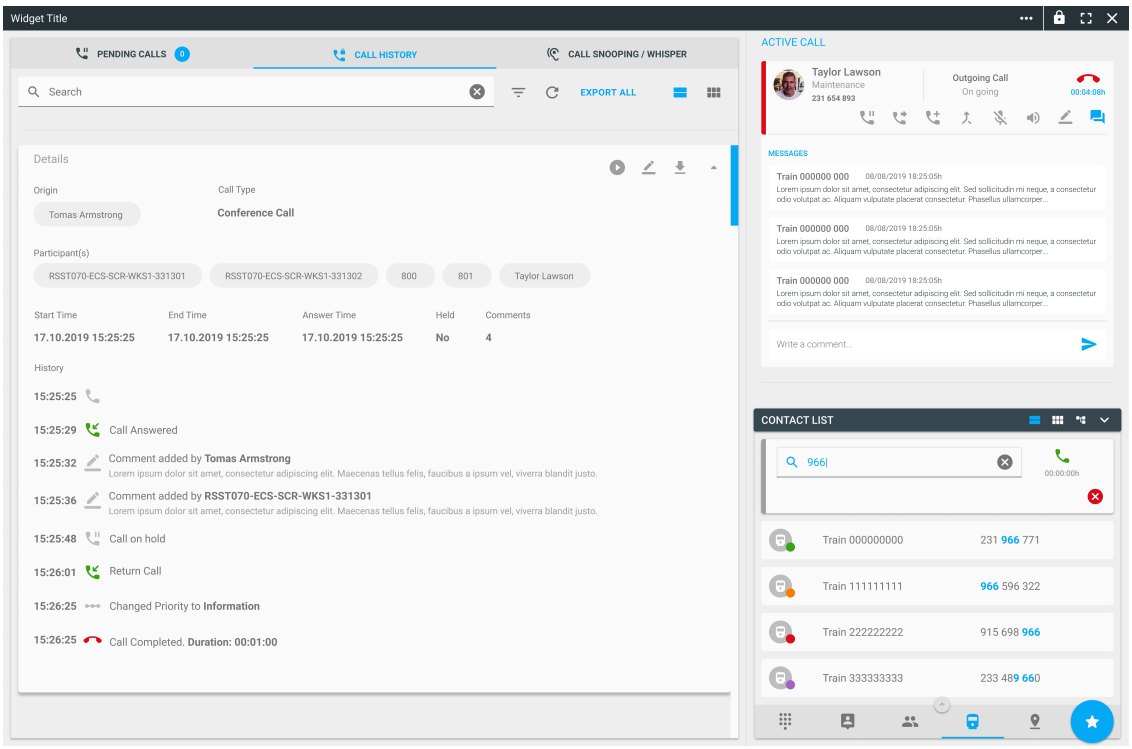


Figure 5.16: Call History - Call Details

In the figure 5.16, we can observe all the information we store regarding the history of a call. One thing in particular that we were required to save was all the operations performed during the call, another one was the ability to recover the recorded call. All this information might end up crucial when analysing the control central performance during emergency calls.

For this release, we completed the following features:

- Expanded Version;
- New Pending Calls Area;
- Call History Area;
- Search on Call History;
- Filters on Call History;
- Refresh Call History List.

Some features initially planned for this release were delayed to the final version, for example:

- Call Snooping;
- Export All for the Call History;
- Playback on recorded Voice Calls.

5.4.2.2 Final Version - Call Manager (Expanded Version)

On this final version, we were tasked with the production of the features Call Snooping and Call Whisper, these were the final steps in the development of this widget, after this, we were ready to pass to the testing team and wait for the release.

To give a brief idea of the requested features, Call Snooping is an operation where the user is able to wire a call from other operators without notifying them that someone joined the call. While Call Whisper starts the same way but also gives the user the possibility to talk (whisper) to one of the operators in the call.

These two features are connected because to be able to perform a call whisper operation, the AD operator would need to execute a call snooping first, as a result of this we chose to put them on the same page.

According to the product requirements, these two features shall only be accessible to operators with the proper role, so we established only supervisors with the right permissions were able to access this page. This is one example of how we made use of the different roles and permissions in the development of our application.

Finally, after these two features were added to the application, it was ready to be released, and this is what the call snooping page looked like:

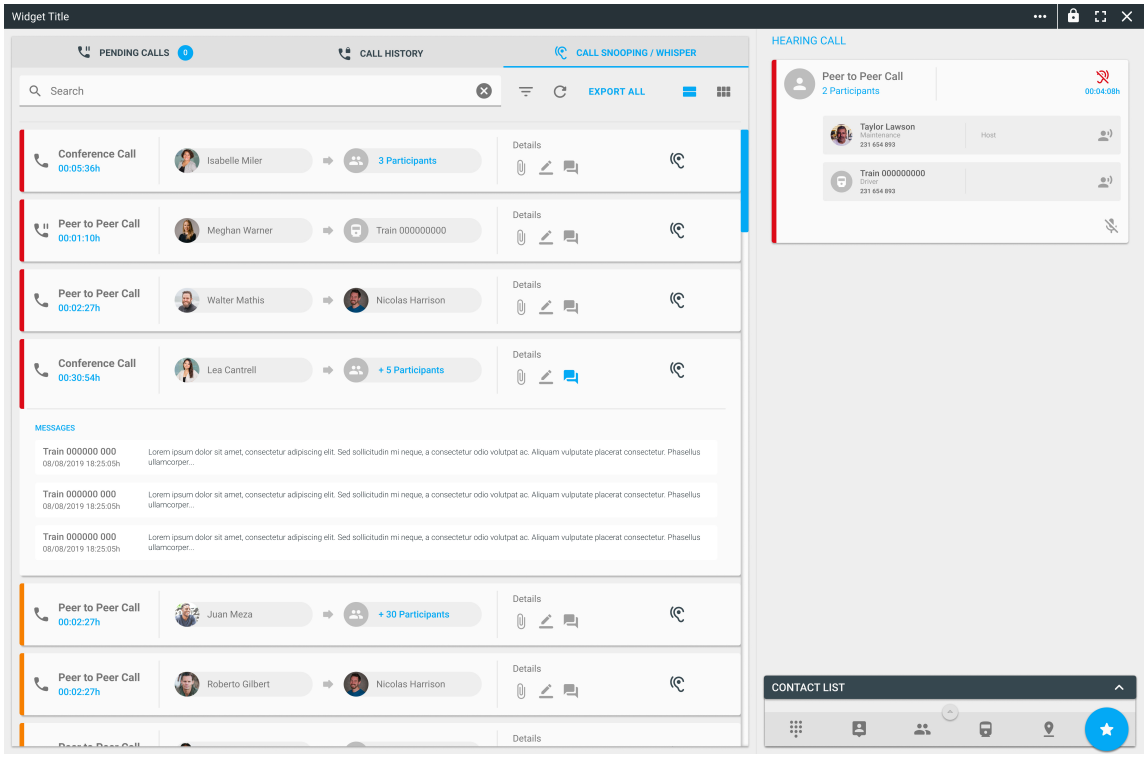


Figure 5.17: Call Snooping/Whisper Page

5.4.3 Contact Manager

This is the other main widget to develop, in this one, the user will be able to manage their contact list, and their filters list, including their favorites. These two features were separated into tabs for a clean and intuitive usage, while also leaving the possibility to combine more features to this widget.

Regarding the interface, due to the fact that this was a management widget, it would be used less frequently than the Call Manager widget and it would not be necessary to use at the same time that other GDP applications. And for this reason, we opted to develop only a maximized version of this widget.

5.4.3.1 Contact List

For the contacts we chose to go with three different types, Simple (trains, firefighters, police, help points, etc), AD User Operator (Role) and Group.

This tab contains all the usual contact management features, for example, add/delete/edit contacts, create/delete groups, add/remove contacts from groups, etc.

For the designs of the contacts, we opted again with the cards, because the user would already be familiarised with the concept of the cards. As we can see in the figure 5.18, we decided to only place the most important information, like the contact photo, name, type and number. The remaining information could be accessed with a click on the card and it would show the details of the contact in the rightmost column, as it shows in figure 5.19.

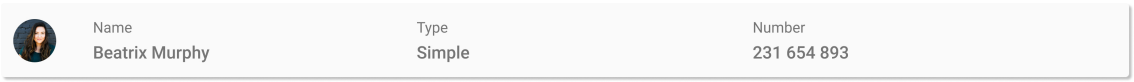


Figure 5.18: Contact Card Design

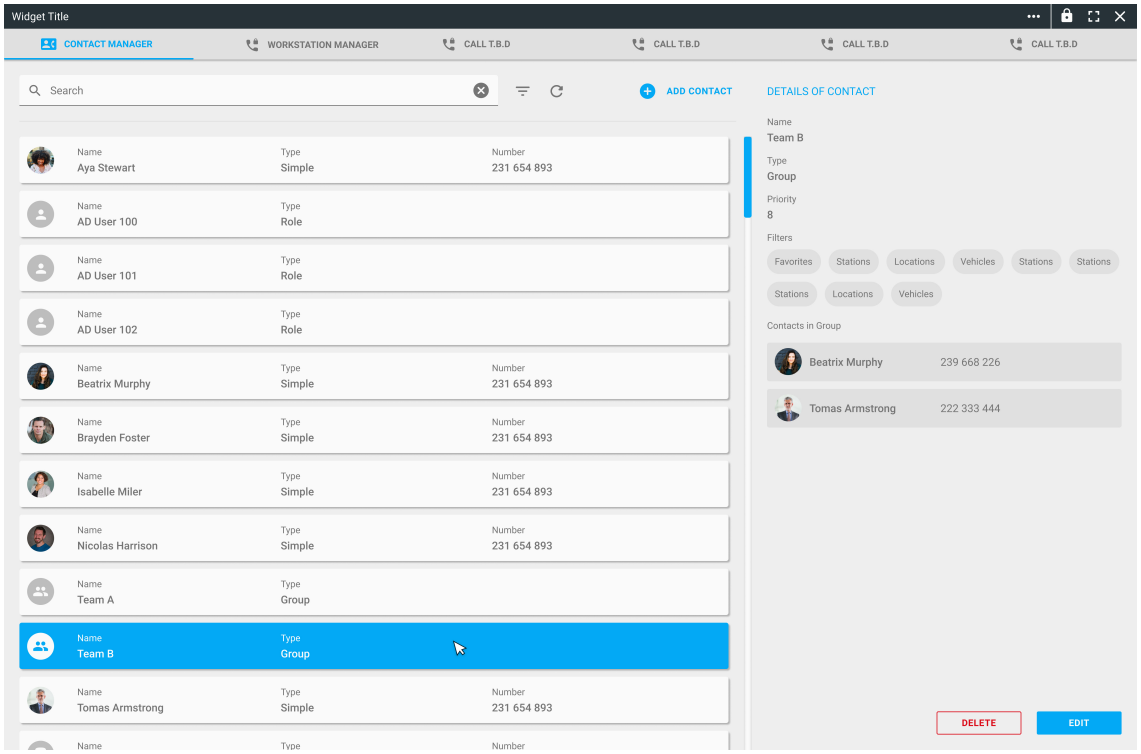


Figure 5.19: Contact Manager - Contact Details

One big challenge we had was how we would implement the adding contacts to a group feature, we ended on two options. The first one, where we would open a new window with the same list of contacts as the main page, but we found this to be a little redundant, so we went with the second option. The second option would end up reusing the list on the main page to select the contacts to add to the group. In the figures 5.20 and 5.21 we can see this process, this can be a little confusing to use at the beginning but in the end, looked way cleaner and without unnecessary windows.

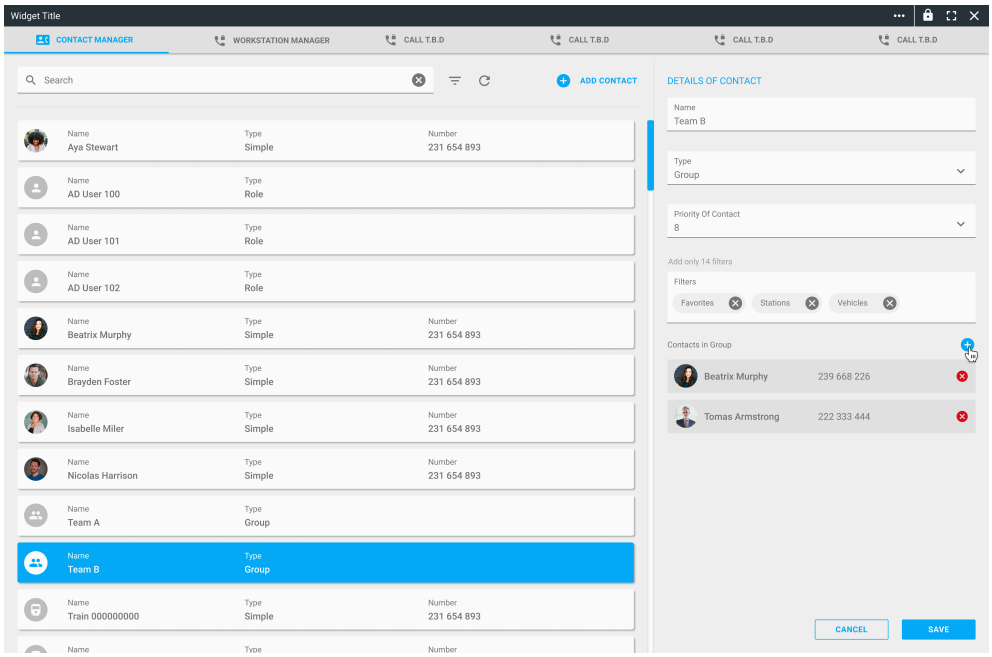


Figure 5.20: Contact Manager - Edit Group Contact

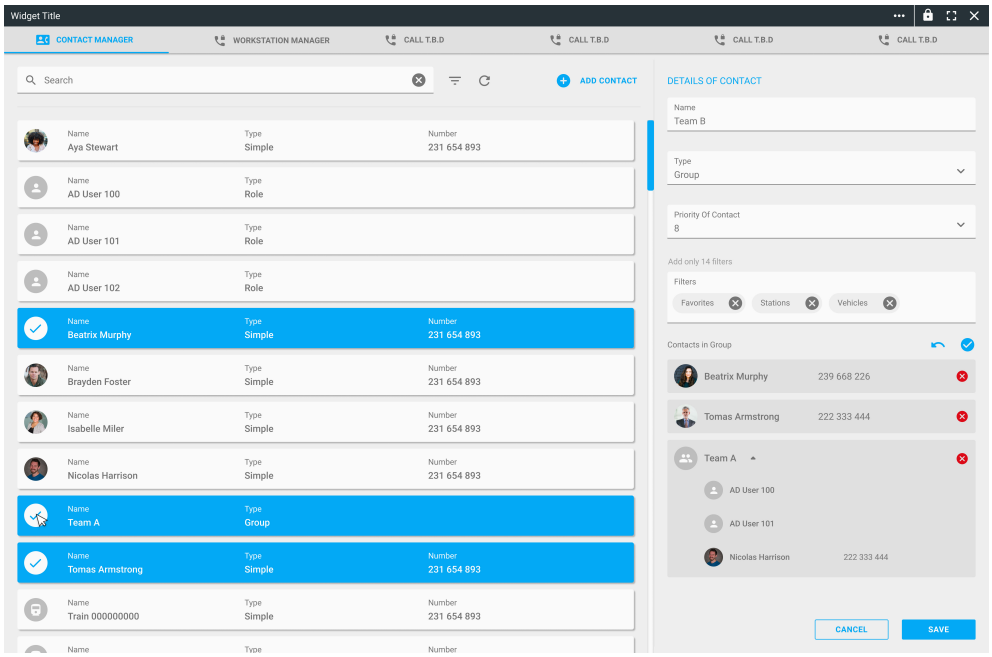


Figure 5.21: Contact Manager - Add Contacts to Group

5.4.3.2 Filter List

To make our application more flexible in its usage we created this section (Fig. 5.22), where the user is able to filter the contacts, for example, by location, by groups, entities or even vehicles types (trains). The user is also given the opportunity to edit their contact list tabs on the call manager widget. With this new functionality, the user can arrange their contacts in their own way.

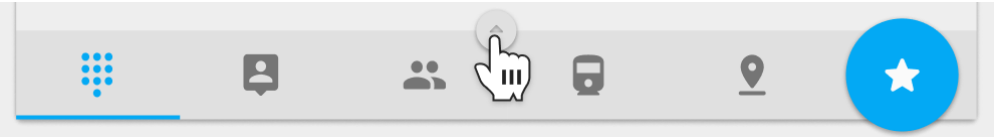


Figure 5.22: Contact List Tabs

As we can see in the figure 5.23, we have a locked filter, the favorites, represented by the blue star, this is also a locked filter in the contact list tabs as we can see in figure 5.22). Here the user can add their favorites contacts to this filter making them easy to access in the call manager app.

When developing this section we faced a similar challenge to the contact manager, about the feature of adding contacts to the filters. We opted to go for the same strategy by replacing the filters list with the contacts list page, this would make the user free of useless opened windows. We can see this procedure of adding contacts to the favorites filter in the figures 5.23 and 5.24.

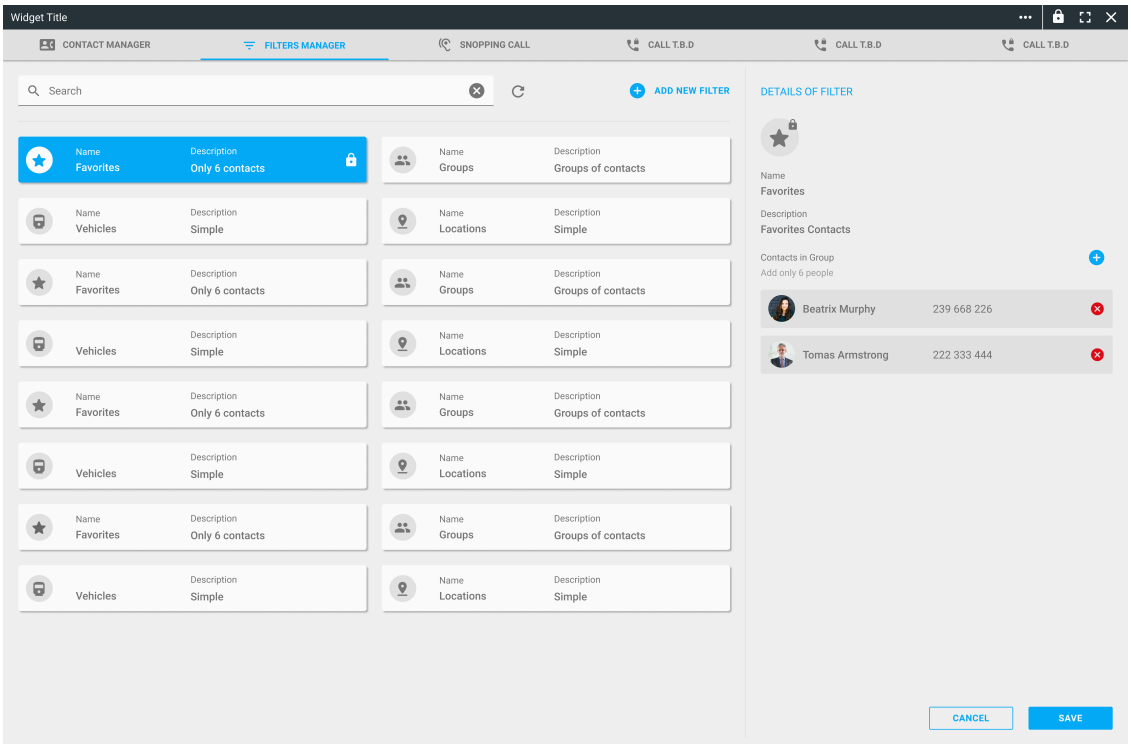


Figure 5.23: Filters Manager - Favorites

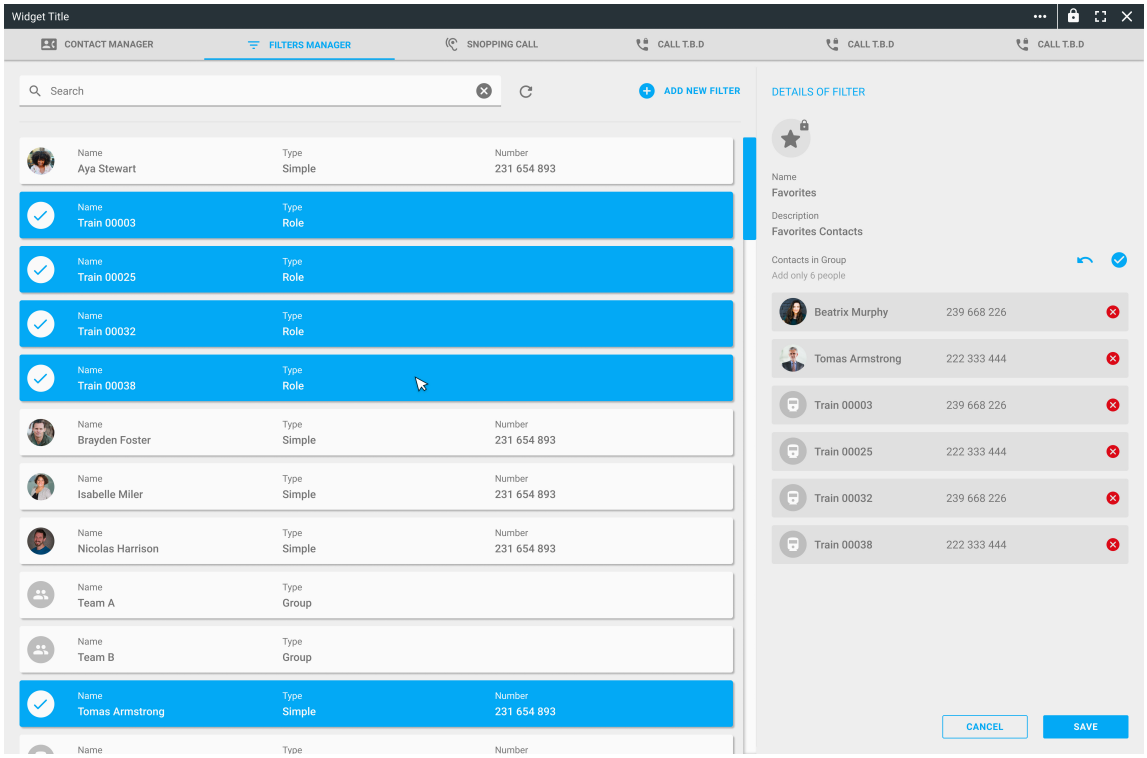


Figure 5.24: Filters Manager - Add Contacts to Favorites

In this Section the user would be able to add, edit or remove filters. When editing a filter, he can change the filter icon, name or description, while also being able to add or remove contacts from their contact list.

After we finished the filters page, the contact manager widget was ready to be tested and released.

Chapter 6

Verification and Validation

In this chapter, the different phases regarding the verification and validation of the project will be described in extensive detail.

When the application was ready to be tested, we came up with the idea of integrating our product's testing in the scrum daily meetings of the Thales teams, including ours, in these meetings we made sure of always having one of the AD developers present in order to streamline these meetings. We would like to thank the different teams from the Thales' product department that were available to get to know our product and help us with its testing.

These dailys helped verify and validate the application's performance in the context of the requirements linked to voice communication.

When we moved to this method, we noticed a significant improvement in the quality of our product, testing and bug fixing became faster and more efficient, this process also included the registration of these bugs on the Jira [2] platform followed by the respective assignment to a developer who proceeded with its resolution.

In the figure 6.1 we can see a screenshot of the Jira platform dedicated to managing the requirements and bugs of our application. In this example, we see an issue assigned to me with the purpose of solving a bug found in the *favourites* functionality in the Call Manager widget. This bug was found and reported by the testing team during the review of Call Manager's first version, where one of the many features submitted was the ability to create calls through the user's favourites.

After fixing this bug I updated the resolution in the platform to Done. Following this process, it is the responsibility of the product manager to report if the issue was indeed completed and change its status to closed. This was an example of how a bug was handled by our team, detailing its entire process in order to give a picture of my responsibilities towards the AD product.

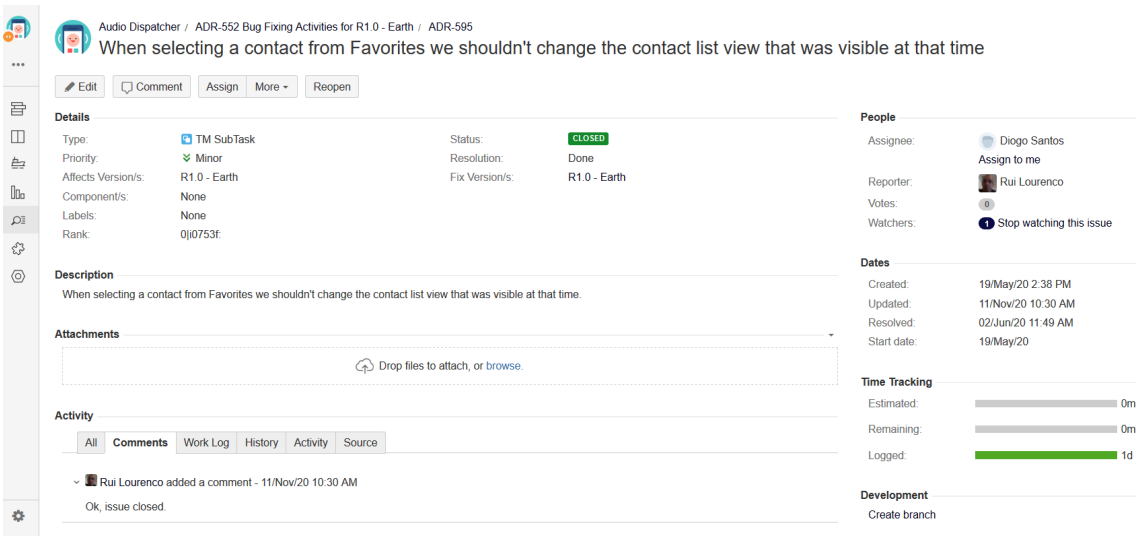


Figure 6.1: Audio Dispatcher Bug Fixing - Jira

To validate our project according to Thales metrics, it was necessary to create and perform unitary tests on the software, then, with the help of the platform SonarQube, inspect the quality of the code in order to have high code coverage in Unit testing.

The figure 6.2 shows one of our application code reviews, this was performed by SonarQube. This platform provides an overview of the overall health of the source code by performing a deep analysis of the quality of the code and submitting the results in the platform dashboard. This analysis was complemented by using JaCoCo. JaCoCo is the toolkit that was used to calculate the Java code coverage and generate detailed reports of our Project's modules.

This section will focus more on the code coverage analysis of the application since this is the best metric to evaluate the quality of the unit tests. According to Thales IVVQ strategy, every product's module shall be evaluated with a result of at least 90% code coverage in unit testing.

Code Coverage is a metric used to check the quality of a project's testing, as it represents the percentage of the production code that has been tested.

In the SonarQube context, it is a mix of the Line coverage and Condition coverage metrics.

Line coverage checks if a given line of code has been executed during the execution of the unit tests, essentially, is the metric that provides the density of covered lines by unit tests. Condition coverage is the density of possible conditions in flow control structures that have been followed during unit tests execution. It measures the proportion of conditions within decision expressions that have been evaluated to both true and false.

According to the Metric Definitions of SonarQube, the Code Coverage is computed as follows:

$$Coverage = (CT + CF + LC) / (2 * B + EL)$$

where

- CT = conditions that have been evaluated to 'true' at least once,
- CF = conditions that have been evaluated to 'false' at least once,
- LC = covered lines = lines to cover - uncovered lines,
- B = total number of conditions,
- EL = total number of executable lines (lines to cover).

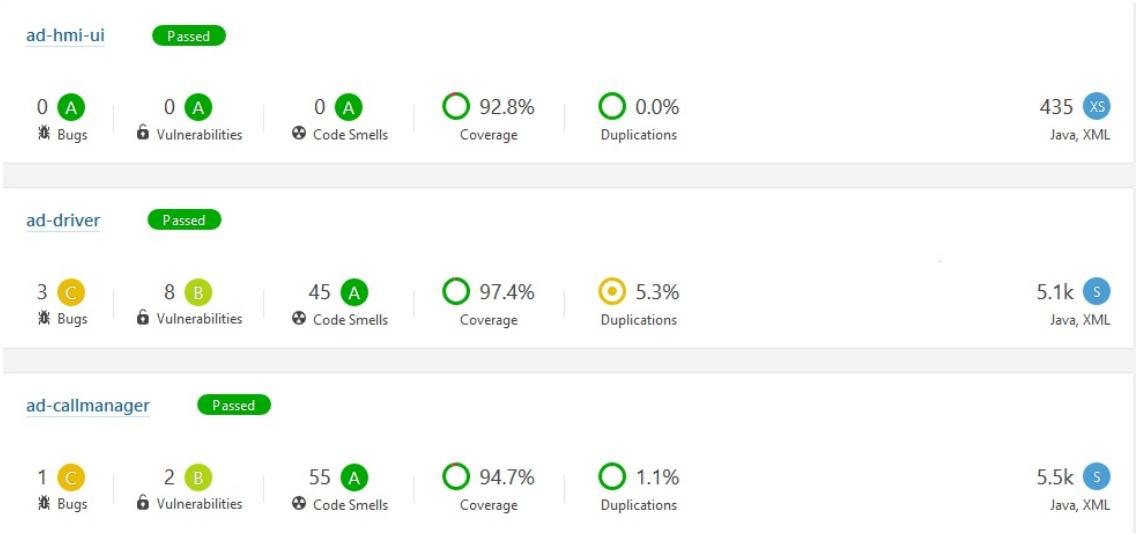


Figure 6.2: Audio Dispatcher Code Review - SonarQube

As we can see in the figure 6.2, there were three modules analysed. Of these three, I was responsible for the *ad-hmi-ui* module displayed in the first line of the image, the other two represent the source code remaining. Before each release, we were tasked with reviewing the source code, every module would need to be within the acceptable parameters. After the conditions were met, we had permission to push the version into production.

During my time at Thales as a developer of the Audio Dispatcher, we had three major releases. In each release, all the system's modules, including the modules I was working on, had to be unit tested on at least 90% of its code.

After this process, we pass the project to the IVVQ team where they formally define all measures needed to be taken to perform the integration, verification, validation and qualification activities for the system. These are the main pillars of the IVVQ Strategy for our product:

- Implement Continuous Integration (enabling Continuous Inspection);
- Implement Continuous Delivery;
- Try to Automate as much as possible (or viable);
- Include cybersecurity penetration tests for every major release;
- All 3^{rd} party suppliers integrations (e.g., PABX, GSM-R, Help Points, SIP converters) to be (re)qualified for each major release;
- Every reusable module shall comply with the following test coverage:
    1. Unit testing: > 90% code coverage.
    2. Integrated tests: Test the entire component. Mocks are allowed for this component dependencies.
    3. UAT (User Acceptance Tests): As a minimum, cover all requirements listed in the SSS [17].
- Provide visibility points (delivery-ready) to the project team every 6 weeks.

The IVVQ team organizes their testing into different documents, the main one being the Software Test Description (STD) [18]. This document provides a detailed description of the test preparations, test cases, entry and exit criteria and procedures to be used to perform Audio Dispatcher qualification testing. This is a living document and is edited and updated to reflect the evolution of diverse aspects of Audio Dispatcher.

Other important documents are the Software Test Reports (STR) [19], which are created after every release for their respective testing. These STRs show that the tests were executed, passed or failed and if the test exit criteria were met. These STRs also reference issues and bugs found during the test campaigns that were listed in a dedicated tab.

All AD Product requirements should be managed by the Product Owner both with DOORS [9] and Jira [2]. They are discussed and addressed in the STD and finally taken by the SCRUM team. Most tests are derived from specified requirements so that the testing activities can show that the product is compliant at an acceptable level.

In Figure 6.3, we can see a test case registered by the IVVQ team and documented on the STD.

AD-STD-051 Make call (P2P)

Outline:	Audio Dispatcher functionality to make a P2P call.
Requirements Tested:	**[AD-STD-REQ-051]** The AD system shall allow the operator to make P2P calls to other operators or external entities.
Prerequisites:	• Audio Dispatcher Core and HMI components are deployed and running. • No incoming calls are coming in the system • The operator's extensions are available and registered on the system
Test Procedure:	
Given the AD Hmi Main Dashboard Page When the user selects the Audio Dispatcher Tab And selects a contact from the contact list to make the call to And clicks the make call button on the call menu	
Expected Result:	
Then the call is created And its ringing on the other peer but not answered	
Notes:	
Test done by inspection.	

Figure 6.3: Test Case Example

In the image below (Fig. 6.4), we are presented with two automated test cases of a test suite containing the Contact Manager widget related tests. These tests were created using the Robot framework with Selenium Webdriver. Robot framework is a test automation framework for acceptance testing and acceptance test-driven development. Selenium Webdriver is used to create test scripts that are used for automating web-based application testing in order to verify that the application performs as expected.

Automation testing is one approach that the IVVQ team chose to follow as it enables the possibility of reusing the same test scripts for different versions of the application. This approach reduces future testing efforts and grants a faster delivery capability.

In each Test case shown in figure 6.4, we can see the actions the user takes in order to reach the desired outcome. Each test will validate the required test case, like the example presented in the image above (Fig. 6.3).

```
*** Settings ***
Documentation     A test suite containing Contact Manager related tests
Suite Setup       Run Keywords    Set Selenium Timeout    60s    AND    Open Web Browser    AND    Set Selenium speed    0.1s
Suite Teardown    Close All Browsers
Test Setup        Run Keywords    Delete All Cookies    Close All Browsers    Open Web Browser
Test Teardown     #Run Keywords
Resource          ../Resources/gherkin_kw.robot
Resource          ../Resources/backend_requests_kw.robot

*** Test Cases ***

AD-STD-040 MANAGE CONTACTS
    [Documentation]  The Contact Manager is successfuly displayed
        Given I login to Audio Dispatcher
        When I am in the Home Page
        And I click on "Widget Button"
        And I click on "Contact Manager Button"
        Then The "Contact Manager" widget is displayed


AD-STD-041 ADD CONTACT
    [Documentation]  Audio Dispatcher functionality to add a contact through the contact manager
        Given I login to Audio Dispatcher
        When I am in the Home Page
        And I click on "Widget Button"
        And I click on "Contact Manager Button"
        And I click on "maximize contact manager"
        And I click on "Add Contact" (use executejavascript)
        And I fill in the new contact information
        And I click on "Contact Save Button" (use executejavascript)
        Then I See "AutomationTester" in the "Contact List"
```

Figure 6.4: Audio Dispatcher - Automation Testing

Before each release, the client-side and server-side of the application needed to be in synch, in such a manner that there were no problems with the features to be delivered. For this reason, additional meetings were held during the weeks of releases in order to optimize communication between the two sides so that there were no mistakes. Afterwards, the product proceeded to the testing stage with the IVVQ team, which implemented the strategy mentioned above. At the end of this process, the version of the product in question was ready.

During the development of the application, several presentations were made to potential clients, during these meetings, ideas and suggestions were exchanged between both parties, in order to reach the same goals in line with the customer needs. We would then register their feedback, and rearrange the product's requirements accordingly. This led to a constant improvement which was possible to demonstrate in the following presentations with these clients.

From these meetings we had with clients, I can highlight one, in which we met with representatives from a railway operations control center in a large European city. During this presentation, we showed a demo of our product, in which we specified all the widgets' features and highlighted the differences that our application had when compared with other products and technologies in the market and its excellent suitability for the railway sector.

One of the clients' concerns was that the application should be as user friendly and intuitive as possible so that the training and adaptation period of the application users would be quick and efficient.

All my work developed in the AD project was validated through several testing processes for each release. In the last release where I was involved, the testing team filled out a Software Test Reports (STR) [19], verifying and validating all the functionalities required in that release. Note that this STR had an area in which were referenced issues and bugs found during the test campaigns, which were pointed as tasks for the next releases.

After this last delivery, the feedback obtained, not only from the testing team but also from all the participants in the AD project, was very positive that allowed me to gain a better perspective and realize that my work will have a significant impact on the future of the application.

Chapter 7

Conclusion

The aim of this dissertation was to implement a brand-new human-machine interface (HMI) for the Audio Dispatcher project, which included the development of two web widgets, namely, *Call Manager* and *Contact Manager*.

This objective was achieved and I have described in this thesis all the aspects that were necessary to successfully accomplish what I set out to do.

In the first chapter, I started by presenting a brief context of what the product is about, followed by the main motivations behind the product's idea and what the vision for this application would be.

In the next chapter, I specified the technical objectives, by explaining what were the main purposes of the Audio Dispatcher Project. To better contextualize these objectives, we began by defining the Roles and profiles of the application, followed by the user's requirements, where we organized in tables the requirements important to my work, whether they are functional or non-functional. Next, we presented two use cases diagrams, to summarize all the features being developed, the first referred to the Call management Widget, while the second applied to the Contact manager Widget. This chapter ended with a brief reference regarding similar products. Here, it was important to understand, that although there are already a few software in the market that allow the unification of various forms of communication in a single application, the fact that the AD is developed exclusively focused on the railway operational centers, it has the possibility of being easily configurable in order to comply with all the requirements that this sector needs to satisfy. This is what makes the AD a unique and differentiated product.

In chapter 3, we researched and defined all the software and technologies used in the project. From the Thales digital platform description, communication interfaces, and tools to validate the application code. In this chapter, we also described the data

management tools, and several communication frameworks.

Chapter 4 resumed the methodologies adopted by the Thales development teams, here we described the Agile and Scrum approaches used in the AD team management. I should mention, the Scrum method used, allowed me to easily adapt, have continuous feedback on my work, and feel a consistent improvement in my performance. These descriptions were then followed by the management tools essential to organize the project's progress.

In the Approach (chapter 5), we start by describing the solution architecture of the whole system, and how all the components are interconnected. This was detailed by providing an explanation of how the communication between modules is performed using events. Furthermore, there is a section specifying the databases in detail, including the data models' entities.

Following the system's specifics, we elaborated a description of the Frontend development. Given that I was the main developer responsible for implementing the interface of the Audio Dispatcher system, this section was more detailed than the other subjects.

The section was divided into three parts, of these, two were for the *Call Manager* widget, (one for the compact version and another for the expanded version). The last part of the section concerned the *Contact Manager* widget. In these parts, we described all the functionalities implemented for each version, while describing some of the challenges encountered. Each part ended with the presentation of the finished product.

In the last chapter of this dissertation, I explained how the testing and validation process of my work went and also described the strategy the IVVQ team followed regarding our project.

The product improved remarkably since the beginning of my work at Thales. When I started, the application was still in the initial phase, as only a few basic call features were being developed at the Backend services.

When I was first presented the project and introduced to the team I was really excited for this opportunity. I had the chance to work in a renowned company, where I had all the conditions needed for the development of this thesis. After the adversities the country faced, with the Covid pandemic, which forced all Thales employees to work from home, Thales' supervisors had an important role in making sure all activities continued as planned. This involved the Audio Dispatcher team, which created new challenges with the development of the product.

The AD team consisted of really professional developers who welcomed me with open arms and included me as an equal since the beginning. It was challenging and exciting to have worked on such an important project where collaboration between everyone was key.

Regarding the proposed features of the work-plan prepared at the beginning, we managed to develop most of them, one to consider is the calls' history interface, which was initially planned, however, we had some doubts about whether we would have the time to implement it or not. Another feature to recognize is the call forwarding strategies, such

as call diversion and hunting mode. These features improved the system's availability, as these methods (algorithms) are used to ensure that the call is answered by finding an available operator.

Some requirements were postponed in regard to the client's needs and Demo presentations, this resulted in some features planned to be developed being left out. In the postponed list, we can highlight features like, the messaging system (chat), calls comments, user's chat history, these features were not developed during my internship but were planned to be developed in the following releases.

One main component that will be added to the next release will be the systems settings manager widget, this was something the clients manifested their interest in, where it would provide the AD supervisors more possibilities regarding Audio dispatcher control and operators' management. This release will also feature the introduction of areas of responsibility, this feature is crucial in the context of the railway operational control centers, where the operators would be able to effectively manage different stations and their respective stakeholders.

Another feature that will strengthen the position of the product in the railway industry is video calls, this one, is something that the AD team certainly wants to integrate into the product. This feature will be vital when competing with the different communication platforms in today's market.

During my time at the company, I took part in many meetings with potential clients, in these meetings we had the chance to present our product. After these presentations, the clients would present their concerns and ideas about our application, we would take these into account and would revise the product's requirements and even make some changes according to the clients' needs. Before I left, many other meetings were already scheduled, with the same clients, as well as potential new clients. The team saw this as an excellent sign for the future of the Project.

Evidently, there is already a long term plan scheduled for this product, this joined with the clients' meetings feedback, we can envision how the project will move forward. It will be a successful path with a lot of potential.

Bibliography

[1] *Asterisk*. URL: https://www.asterisk.org/ (visited on 05/16/2021).

[2] Atlassian. *Jira Software*. URL: https://www.atlassian.com/software/jira/ (visited on 05/27/2021).

[3] J. Brito and R. Lourenço. *Solution System/Subsystem Design Description (SSDD) – Audio Dispatcher 3*. 2019.

[4] *Cloud Foundry*. URL: https://www.cloudfoundry.org/ (visited on 05/16/2021).

[5] I. Fette and A. Melnikov. *The websocket protocol*. 2011. (Visited on 06/15/2021).

[6] *Gerrit*. URL: https://www.gerritcodereview.com/ (visited on 04/21/2021).

[7] *Git*. URL: https://git-scm.com/ (visited on 04/21/2021).

[8] Google. *Polymer*. URL: https://www.polymer-project.org/ (visited on 05/16/2021).

[9] IBM. *IBM Engineering Requirements Management DOORS Family*. URL: https://www.ibm.com/products/requirements-management (visited on 05/27/2021).

[10] *Jenkins*. URL: https://www.jenkins.io/ (visited on 05/27/2021).

[11] *Keycloak*. URL: https://www.keycloak.org/ (visited on 06/12/2021).

[12] R. Lourenço. *System Engineering Management Plan (SEMP) – Audio Dispatcher 3*. 2019.

[13] *PostgreSQL*. URL: https://www.postgresql.org/ (visited on 06/12/2021).

[14] *RabbitMQ*. URL: https://www.rabbitmq.com/ (visited on 06/12/2021).

[15] *Redis*. URL: https://redis.io/ (visited on 06/12/2021).

[16] R. Ribeiro and R. Lourenço. *Interface Control Document (ICD) – Audio Dispatcher 3*. 2019.

[17] R. Ribeiro and R. Lourenço. *System/Segment Specification (SSS) – Audio Dispatcher 3*. 2019.

[18] *Software Test Description (STD) – Audio Dispatcher 3*. 2020.

[19] *Software Test Report (STR) – Audio Dispatcher 3*. 2020.

[20] *SonarQube*. URL: https://www.sonarqube.org/ (visited on 06/15/2021).

[21] A. Tiplé, J.-C. Menchi, and J. L. Arana. *Thales Ground Transportation Digital Platform*. 2016.

[22] A. Veiga. *GDP UI Toolkit v2 - GTS Software Engineering*. 2019.