



## **Bot Federation**

# Skill Delegation Feature for Wit Bot Engine

Master's degree in computer engineering – Mobile Computing

João Paulo de Oliveira Sismeiro

Leiria, September 2019

*This page was intentionally left blank*



## **Bot Federation**

# Skill Delegation Feature for Wit Bot Engine

Master's degree in computer engineering – Mobile Computing

João Paulo de Oliveira Sismeiro

Internship Report under the supervision of Professor José Carlos Bregieiro Ribeiro, and  
Pedro Andrade

Leiria, September 2019



*This page was intentionally left blank*

# Abstract

---

Messaging applications are used on a daily basis for communication between people around the globe. Even though the main purpose of this type of application is to provide a service where people can join in order to interact with one another, it has evolved since and automated conversation has grown. This type of automated conversation is only possible through the implementation of intelligent agents called chatbots. The chatbots objective is to mimic user behavior in order to understand and reply to a given user input. This project aims to further improve the already existing mechanisms for this chatbots so that they can interact with other chatbots so that users can, in a single conversation, have access to multiple chatbots. This is possible through the implementation of a series of features that allow chatbot creators to create groups of chatbots, called federations, and delegate the conversation from one chatbot to another chatbot available in a federation. This approach aims to reduce the number of failed responses and at the same time prevent the user's frustration because if a chatbot cannot answer, the user needs to find and open a conversation with a chatbot that can answer.

Every chatbot has a series of tasks and knowledge that he can use to answer user questions, but there is always a reduced scope so that the chatbot can perform at his best. This solution's objective is to keep this approach of reduced scopes for better performance, but every chatbot can contribute with his knowledge by lending his services to another chatbot.

Keywords: Chatbot, Messaging Applications, Automated Conversations

*This page was intentionally left blank*

# List of figures

---

- Figure 1 - The tasks of NLP and NLU (adapted from [4]) ..... 6
- Figure 2 - Most important chatbot concepts (adapted from [6])..... 9
- Figure 3 - Project Schedule diagram..... 22
- Figure 4 - Bot creation dialog ..... 26
- Figure 5 - Bot conversation builder ..... 26
- Figure 6 - Bot builder components ..... 27
- Figure 7 - Architecture diagram..... 34
- Figure 8 - Skill Mapping Interface ..... 37
- Figure 9 - Create/Edit Skill Mapping Dialog..... 38
- Figure 10 - Bot Control Panel..... 39
- Figure 11 - Federation Channel Configuration Interface..... 40
- Figure 12 - User texting chatbot with federation enabled ..... 49
- Figure 13 - Music bot default answer ..... 50
- Figure 14 – Delegation process finished..... 51



*This page was intentionally left blank*

# List of tables

---

Table 1 - Project Use Cases and priorities ..... 19

*This page was intentionally left blank*

# List of acronyms

---

- AI** – Artificial Intelligence
- AIML** – Artificial Intelligence Markup Language
- API** – Application Programming Interface
- ASR** – Automatic Speech Recognition
- AVS** – Alexa Voice Service
- BPMN** – Business Process Model and Notation
- IDE** – Integrated Development Environment
- NLP** – Natural Language Processing
- NLU** – Natural Language Understanding
- RCS** – Rich Communication Service
- REST** – Representational State Transfer
- RPC** – Remote Procedure Calls
- SDK** – Software Development Kit
- SIML** - Synthetic Intelligence Markup Language
- STT** – Speech-to-Text
- TTS** – Text-to-Speech
- UI** – User Interface
- XML** - Extensible Markup Language

*This page was intentionally left blank*

# Table of Contents

---

- ABSTRACT.....II
- LIST OF FIGURES ..... IV
- LIST OF TABLES ..... VI
- LIST OF ACRONYMS ..... VIII
- TABLE OF CONTENTS .....X
- 1. INTRODUCTION.....1**
  - 1.1. MOTIVATION..... 1
  - 1.2. OBJECTIVES AND CONTRIBUTION ..... 2
  - 1.3. DESCRIPTION OF THE ADOPTIVE ENTITY ..... 2
  - 1.4. DOCUMENT STRUCTURE ..... 3
- 2. BACKGROUND.....4**
  - 2.1. INTELLIGENT AGENT.....4
    - 2.1.1. *Chatbot* ..... 5
  - 2.2. NATURAL LANGUAGE PROCESSING.....5
  - 2.3. CHATBOT CONCEPTS .....7
    - 2.3.1. *Utterance* ..... 7
    - 2.3.2. *Intent*..... 8
    - 2.3.3. *Entity*..... 8
    - 2.3.4. *Channel* ..... 10
  - 2.4. ARTIFICIAL MARKUP LANGUAGE..... 10
  - 2.5. SPEECH-TO-TEXT AND TEXT-TO-SPEECH..... 11
  - 2.6. ARTIFICIAL INTELLIGENCE ..... 12
- 3. STATE OF THE ART.....13**
  - 3.1. VIRTUAL ASSISTANTS..... 13
  - 3.2. MESSAGING PLATFORMS ..... 15
  - 3.3. CHATBOTS PLATFORMS..... 16
  - 3.4. CHATBOTS LANGUAGES ..... 17
- 4. REQUIREMENTS AND FUNCTIONALITIES ..... 19**
- 5. PLANNING AND METHODOLOGY ..... 22**
- 6. TECHNOLOGIES AND TOOLS..... 25**
  - 6.1. WIT BOT ENGINE ..... 25

6.2.	ANGULAR .....	29
6.3.	JAVA/SPRING BOOT .....	30
6.4.	EUREKA .....	30
6.5.	GRPC.....	31
<b>7.</b>	<b>DEVELOPMENT .....</b>	<b>32</b>
7.1.	DECISIONS MADE .....	32
7.2.	ARCHITECTURE .....	34
7.3.	IMPLEMENTATION.....	35
7.4.	BACKOFFICE.....	37
7.5.	BACKEND SERVICES .....	40
7.5.1.	<i>Administration Service</i> .....	41
7.5.2.	<i>Runtime Service</i> .....	41
7.5.3.	<i>Federation Service</i> .....	42
7.5.4.	<i>Gateway Service</i> .....	44
7.6.	FEDERATION CHANNEL .....	44
7.7.	CONTEXT VARIABLES .....	45
7.8.	OPTIMIZATIONS.....	46
7.9.	FACEBOOK PERSONAS.....	47
<b>8.</b>	<b>DEMO .....</b>	<b>49</b>
<b>9.</b>	<b>CONCLUSION AND FUTURE WORK.....</b>	<b>52</b>
	<b>REFERENCES.....</b>	<b>54</b>
	<b>APPENDICES .....</b>	<b>58</b>





# 1. Introduction

---

Nowadays, people use smartphones and computers more than they used to, and in the United States, 91% of the time is spent using messaging applications [1]. Taking this into account, the number of chatbots that can interact with users using natural language is also rising. These agents are being created to answer many of the user needs. They can do simple tasks, like small talk, or more complex ones such as booking a flight. Moreover, they are often used to act as customer support in many companies, in order to, at least, answer frequently asked questions without the need for human assistance. Even though chatbots have evolved, in order to be efficient in what they do, the scope of the knowledge of these chatbots has been reduced. With the scope reduced, chatbots became more efficient in what they do because they are specialized in that specific topic, but this leads to a new problem: with specialized agents, users need to open multiple conversations in order to have access to more information.

One solution to this problem is enabling communication between chatbots. Communication between chatbots is like “opening a door” to the possibility of using other chatbots as an outsourcing entity and makes it possible to delegate responsibilities to other agents when the one that the user is interacting with cannot answer. The communication between chatbots allows the redirection of the user messages to an agent that actually can answer them. This solution has been discussed before but in fact, there is a market gap that can be explored because there is a lack of solutions like the one proposed. Having this solution implemented will allow the users to have access to specialized agents without the need of opening multiple conversations, which would not only solve the reduced knowledge of each agent that often result in unanswered questions, but also keeping it user-friendly.

## 1.1. Motivation

---

The market of chatbots is evolving at a fast rate and this is a topic that has been discussed lately. Facebook, in a conference, showed that the number of chatbots published on the messenger platform has increased and there are three times more active chatbots in

2018 than the year before and eight billion messages have been exchanged between customers and businesses [2]. Oracle carried out a survey that includes responses from 800 decision-makers such as chief marketing officers, chief strategy offices, senior marketers and senior sales executives from France, the Netherlands, South Africa, and the UK; the survey states that 80% of the respondents said that they already use or planned to use chatbots by 2020 [3]. This great interest in chatbots by the industry shows that the number of available agents should increase, meaning that users will have to interact with more and more agents in the future. The fact that the communication between users and chatbots, at some point, is almost inevitable, means that something must be done in order to please the users so they can get the best experience possible and prove that chatbots can be useful.

The fact that chatbots are directly related to the Artificial Intelligence (AI) area is another reason to invest in this area. AI has been, over the last years, a hot topic hence it has evolved substantially lately. Since there is a considerable amount of research on this area it also means that there is still plenty that can be done and improved with AI-based systems.

## **1.2. Objectives and Contribution**

---

The main objective of this project is to have all the benefits of the specialized chatbots that already exist without the need of having multiple conversation windows open.

This project will not only keep conversations between agents and users more user friendly, because users do not need to manage multiple conversation windows, but it will also make it more enjoyable because when a bot fails to answer a question, the user will be redirected to a chatbot that can answer the question reducing the number of default answers.

## **1.3. Description of the Adoptive Entity**

---

This project was developed in the context of an internship in Wit Software, where the internship was carried out for nine months at Leiria's office. The project was developed to be part of the Wit Bot Engine platform. The Wit Bot Engine platform is a platform, that is

being developed by Wit Software, that offers a series of tools which allow the creation, configuration and deployment of chatbots for multiple messaging platforms.

Wit Software is a company that was founded in 2001 as a spin-off on the University of Coimbra. Wit Software is a company that focuses on the creation of white-labeled products for the telecommunication industry. The company is responsible for developing rich communication services that are used by several telecommunication companies and it is available for many platforms like android, iOS, Web, etc.

## 1.4. Document Structure

---

The rest of this document is structured as follows:

- In chapter 2 provides background by introducing some concepts related to the area of research related to the internship project.
- Chapter 3 discusses related work like frameworks and platforms.
- In Chapter 4 is where the functionalities and requirements of the project will be introduced.
- In Chapter 5 the planning and methodology of this project will be explained.
- Chapter 6 is where all tools and technologies used during the development of this project will be shown and explained.
- Chapter 7 details the development process by explaining the changes in the already existing components of the Wit Bot Engine platform as well as the new components created for this internship project.
- Chapter 8 shows a demo from the perspective of a user that talks with a chatbot after all the necessary configurations.
- Chapter 9 is where the conclusions and future work will be detailed.

## 2. Background

---

The concepts that will be explained in the subsections below will be related to the chatbot area and adjacent areas. The aim of those subsections is to introduce the area in a practical way by providing examples.

### 2.1. Intelligent Agent

---

Intelligent agents are autonomous applications that use AI in order to be able to successfully complete the tasks they are conceived for. Agents are often related to the AI area because, in order to have an autonomous behavior, AI is important. Even though AI is important, it does not mean it is always necessary because most simple behaviors can be achieved without AI. For example, if a chatbot has only simple commands like “/help”, “/FAQ” or “/quit” it is easy to map an action because there are no entities to extract and the user intention is clear.

Since AI is a hot topic and has evolved substantially lately, this specific area of intelligent agents has also grown lately and over the years there has been an improvement of what this intelligent agent can do. Intelligent agents usually have a perception of the surrounding environment that can change the way they behave. Since not every intelligent agent has the perception of the surrounding environment, they usually get the context from the user’s input.

There are many types and many applications for intelligent agents; the study case of this project is chatbots – that can be considered to be intelligent agents.

### **2.1.1. Chatbot**

---

Chatbots share plenty with the concept of intelligent agents but are specifically designed to be used through messaging applications such as Facebook Messenger, Telegram, etc. These messaging applications are often called communications channels for chatbots because they are responsible for the exchange of messages between the user and the chatbots.

This type of intelligent agents is known to receive inputs from users in text format and they use Natural Language Processing (NLP) to process the received messages. Once the message is processed and the user intention is known, they can autonomously send the appropriate response back. Since at the present time chatbots use AI to be able to answer a user question, and since it is subject to fail, one common solution is the implementation of default answers that are sent to the user when the system fails to find the user intent.

Chatbots are commonly used to be the entry point for users that want some quick info about a product or a business, but chatbots can be used to do much more than answer frequently asked questions. One example of what can be done nowadays with chatbots is the Pizza Hut chatbot where the user can, using natural language, order a pizza as they would do in a Pizza Hut store. Currently, it is possible to have complex conversation flows in order to be able to have complex tasks that chatbots can do. Usually, to achieve the best results possible, by using all the necessary mechanisms, it is important to train the chatbots because doing so will improve the chatbot capabilities. This training may not be necessary for simple chatbots that do not use AI.

Nowadays, the most used messaging platforms have a series of tools and APIs available that can be used to connect chatbots with the platform so the user can talk with chatbots through these messaging platforms.

## **2.2. Natural Language Processing**

---

Communication between Humans is important but, with the appearance of the first computers, the communication between Humans and machines started to grow and become

as important as the communication between humans. That growing lead to the creation of NLP because, for humans, using natural language to communicate with machines would be simpler.

NLP is a set of processes that used together can digest a text written in natural language and create structured data. The structured data will help the machine to understand the meaning and the intention of the processed text. Knowing the intention and meaning the machine can act accordingly.

NLP can also differ from system to system because some may receive the input in text and others may receive the input in audio format. The fact that the input is audio will require the NLP process to make an extra step to convert the audio into text. Associated to NLP comes the Natural Language Understanding (NLU) that is a subset of the NLP process. While NLP is often mistaken with NLU they have some differences because NLU has a narrower purpose: on the NLP process, the text is parsed at a syntactic level while on the NLU it will be parsed at a semantic level. This is an important difference because the syntax is more about the structure of the text while the semantic is all about the meaning of it. Having that in mind, NLU is what can really transform a sentence into an intention that can be used as a trigger to a specific action. The process of NLP is more complete when associated with NLU because, even though NLP can be used to digest natural language text, it is NLU that can really extract the meaning.

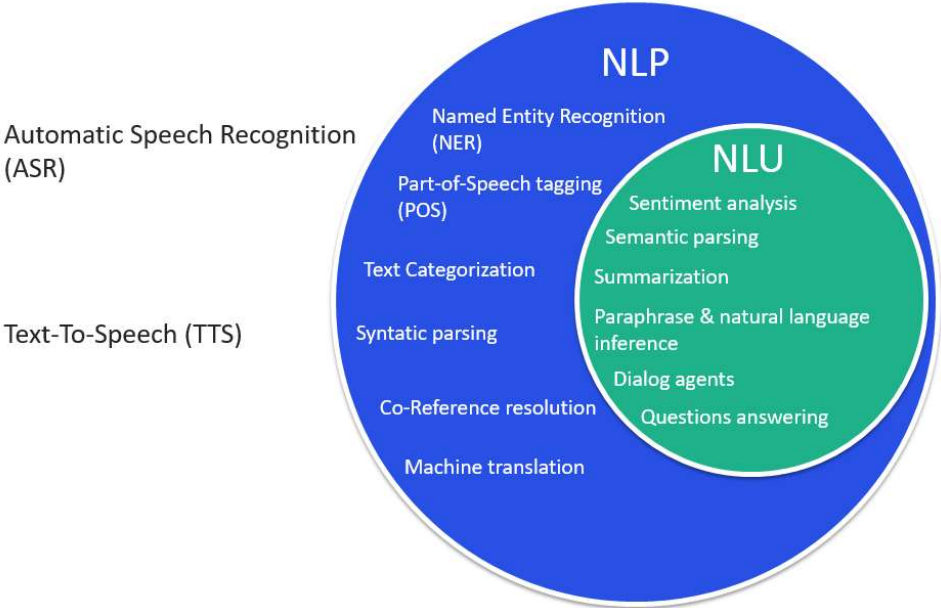


Figure 1 - The tasks of NLP and NLU (adapted from [4])

To better understand the tasks that each process performs and how they are related to each other, Figure 1 shows a diagram that has the most important tasks performed by each process [4].

In summary, NLP is an area of research and application that explores how a computer can be used to understand and manipulate natural language text or speech and act accordingly [5]. The NLU process is what translates user input into a machine-readable structure in a way that makes it possible to extract the meaning behind the user input.

## 2.3. Chatbot Concepts

---

Chatbots have several concepts that make them unique but there are three that when doing some research about them, they always appear. The concepts that are most commonly seen when searching for chatbots are *utterance*, *intent*, *entity*, and *channel*. Those three concepts are going to be described in the following subsections.

### 2.3.1. Utterance

---

The word utterance is, in a simple explanation, the user input, which means that everything that a user sends to a chatbot is an utterance. For example, if a user says something like “I want to book a flight from Lisbon to Paris”, the whole sentence is considered the utterance. In order to train the system to better map a user input into an intent, several utterances are used to increase the system accuracy because there are multiple sentences that can generate the same output. For example, if a user says “Search for an iPhone” or if he says “Find me an iPhone” the intention behind the two sentences is the same hence the output of these two queries will also be the same.

### **2.3.2. Intent**

---

The NLP process has several concepts associated to describe either a step of the process or a result of a specific sub-process. The intent is one of the basic concepts associated with the NLP process and it is always present in an NLP system because getting the user intention is an essential step in order to be able to automate a chatbot that can answer successfully the user requests.

The intent is often described by the NLP systems as the mechanism that can understand and successfully associate the user input with what the user wants. What the user wants can also be described as the user intention and that is why it is called intent. When a user sends a text to a chatbot, the message is processed by the NLP system and one of the steps of this process is responsible to evaluate the user input and associate that user input with an intent. The fact that the NLP system does the mapping between user input and intent makes it possible to create a system that can perform a series of actions in order to fulfill the user's intention. One example of this is when a user sends a message to a chatbot saying that he wants to book a flight from Lisbon to Paris. The NLP process can infer the user intention that in this case, is to book a flight and map it to an intent that can be used by the chatbot in order to successfully do what the user wants.

### **2.3.3. Entity**

---

Entities are the data, mentioned by the users during the conversation, which can be extracted from a sentence. Using the example given the subsection before, when a user asks the system to book a flight, he says that he wants it to be from Lisbon to Paris. In that case, Lisbon and Paris are entities extracted from the sentence and for the NLP system they are considered location entities. Some NLP systems have built-in entities such as date, time, money, location, and many others. With entities, the same can be done with the same intention but with different data that will understand. Returning to the previous example, if instead of booking a flight from Lisbon to Paris the user wants it to be from London to Barcelona, the system will return the same intent but the extracted entities would have



different data. Entities are important to complement the user intention because if he wants to book a flight, as said in the example before, the locations are the required data to book the flight.

Since there are some use cases that cannot be achieved using the default entities, usually, in an NLP system it is possible to create custom entities. This means that in most of the NLP systems there are two types of entities, the default system entities, and the custom entities.

Figure 2 is a visual representation of the three most important concepts related to chatbots [6];

- Utterance – An utterance is anything the user says. If a user types “Book a flight from Paris to Barcelona” the whole sentence is an utterance.
- Intent – An intent is the user’s intention. Intents are given a name usually a verb and a noun such as “bookFlight”.
- Entity – An entity modifies the intent. Entities are sometimes described as slots. Entities are also given a name such as “DateTime”.



Figure 2 - Most important chatbot concepts (adapted from [6])

## 2.3.4. Channel

---

When the word “channel” is used in the chatbot context it refers to conversation channels that are used to act as a bridge between the user and the chatbot. A conversation channel is where the user can interact with chatbot and in the market, and there are several channels that allow this connection between the user and chatbots. One conversation channel that users know and use currently is Facebook Messenger, which is one of the many examples that can surge when talking about chatbots and how users can interact with them.

The channels usually can be separated into two types depending on the way the user interacts. There are channels such as the Google Assistant or Alexa that the interaction between the user and the chatbot is done using voice inputs and on the other side there are channels, such as Facebook Messenger, in which the interaction between the user and the chatbot is mainly done using text inputs but that has also several components that can also be used to interact with the chatbot such as buttons.

## 2.4. Artificial Markup Language

---

Artificial Markup Language (AIML) is a markup language based on the Extensible Markup Language (XML) and it is used to input knowledge into chatbots. AIML was initially developed by Richard S. Wallace to be used to input knowledge into a chatbot called Artificial Linguistic Internet Computer Entity (A.L.I.C.E).

There are several tags on the AIML that can be used to describe the chatbot behavior. The root element of an AIML file is the *aiml* tag and there is only one per file; it is inside the *aiml* opening and closing tag that all the other elements should be placed [7]. The second most important element is the *category* element that is a composition of two other elements called pattern and template. The *pattern* element is used to describe a question and can have words, spaces and wildcard symbols and the *template* element is used to describe the answer to the question described in the pattern element. Category elements can also have elements and tags that are responsible to store some context that can be used in many cases.

The category, the pattern and the template elements are the three basic elements of an AIML file and even though no other elements are necessary to create a basic behavior there are several other elements available that are used to add complexity to what a chatbot can do.

There are some limitations regarding the category element and in an AIML file: a category can only have a pattern and a template element. In order to describe a more complex behavior using an AIML file, it is possible to add more than one category element. There are also other elements, like the *srai* element, that can be used to add more complex behaviors. The *srai* element can be used to simplify a complex grammatical form, reduce many individual sentences into two or more sub-sentences, handle similar sentences, fix spelling mistakes, handle conditional branches, etc. The *srai* element is not mandatory in an AIML file but is important because it will be helpful to create a smarter chatbot.

AIML is now on the version 2.0 and has since changed because there are some new elements, but the core elements are the same and for someone that was already familiar with AIML 1.0 can easily use the new version.

Even though AIML is the standard when talking about chatbot development it is also important to mention a few other alternatives that people can find upon searching about AIML itself. Since they are alternatives to AIML and will not add context value, they will be described in the state-of-the-art section of this report.

## 2.5. Speech-to-Text and Text-to-Speech

---

There are some chatbots in which the interaction with the user is done using voice inputs and for that, it is necessary to a Speech-to-Text (STT) engine and a Text-to-Speech (TTS) engine. These two components are responsible to handle the voice inputs and usually, they complement each other. The STT engine, also called Automatic Speech Recognition (ASR), is responsible for receiving voice input and transforming that voice input into the text because the text is easier to process and is, in terms of memory, lighter than an audio file. Since the response is also in text format a second engine that is responsible to transform the text into an audio format is necessary because the purpose of this type of chatbots is to

have a conversation with full audio input and output to be an experience close to human interaction.

The STT and TTS are not mandatory in every chatbot but is considered to be part of the NLP process because is used in many cases especially on virtual assistants that are designed to interact with humans using voice commands.

## 2.6. Artificial Intelligence

---

AI has become a buzzword in the chatbot industry. Artificial intelligence is concerned with the development of computers able to engage in human-like thought processes such as learning, reasoning, and self-correction [8]. This are key ingredients that chatbots need so that they are able to successfully achieve its purposes. AI is related to the chatbot area because, in order to orchestrate a conversation, even the simplest one, using natural language is necessary to create mechanisms that try to mimic the user's behavior. Chatbots employ AI in many steps of the process, but the NLP process is where the usage of AI is more evident. Since NLP deals with giving the machine human-like capabilities used to understand natural language that is all the languages spoken by humans.

Chatbots use machine learning that is an area of research in AI, to be able to train the chatbot behavior and improve over time.

## 3. State of the Art

---

Before development process begins, it is important to search for platforms, tools, and APIs available on the market that are somehow related to the project in order to have a solid base to start the development. With that solid base, it is possible to acquire knowledge that is important to better understand all the concepts that are somehow related and what can be used to improve the project. Not only it brings useful information for the development process, but it can also reveal some useful tools that can actually be integrated and used for the project itself.

In this chapter, some platforms, tools, and APIs about messaging platforms, chatbot platforms and everything related to the main topic that is chatbots in general.

### 3.1. Virtual Assistants

---

Virtual assistants are, in many aspects, similar to chatbots because in a certain way the objective is to interact with the user using natural language and complete the task sending the result back to the user. Despite having similarities there are many differences between these two because virtual assistants are user-oriented, have bigger scopes and the development is most of the time more complex. The complexity is directly related to the scope because this bigger scope will require a training model that can add more knowledge but still have good results. and the interaction is done using mainly audio inputs. On the other hand, chatbots are business-oriented, have a smaller scope and the development is easier. Having that in mind virtual assistants are often used to, for example, ask the weather, create reminders or ask for some jokes, while chatbots are often used to get customer support like asking for help about a specific product or ask for the list of products. Since virtual assistant's development is complex there are only a few big companies that invest in the development of virtual assistants and succeed, such as Google, Amazon, and Apple.

Google Assistant [9] is a virtual assistant developed by Google that has plenty of built-in tasks that he can complete. Using the Google Assistant SDK [10] it is possible to interact

with Google Assistant without the need of having a Google Home device. The interaction between the user and the Google Assistant is done using natural language and that input can either be text or audio. The input is processed by Google Assistant and a response is delivered. The response can either be in text-only or in HTML. With the HTML response, it comes with much more information and also has access to complementary information such as images, links, etc. The HTML response is important because there is some information that is not possible to have on the text response.

There is also a virtual assistant developed by Amazon that is called Alexa [11] and it is the virtual assistant available on the echo and dot devices. It is also possible to communicate with Alexa using the Alexa Voice Service (AVS) [12] that is a service that only accepts audio inputs because internally the system uses a speech recognition engine in order to transform the audio into text.

On the Apple side, Siri [13] is a well-known virtual assistant used on Apple products such as iPhones, iPads, and MacBook. The fact that Apple restricts the access of many of its services to Apple products makes it impossible to access Siri with products that are not created by Apple.

Companies such as Samsung, Microsoft, and Xiaomi have also taken some steps towards the creation of proprietary virtual assistants, but they do not have the number of features or maturity to be considered direct alternatives to Google Assistant and Alexa, that have been on the market for years.

As for the open-source alternatives, there is Snips [14] that is an open-source virtual assistant that has plenty to offer to the users. Snips, besides being completely open-source, is a virtual assistant that does the processing on the device, works offline and is completely customizable. Snips has a platform where it is possible to create new capabilities for snips that can be created and trained using an NLU system available on the platform. There is also the possibility to choose capabilities created by other developers and there is already a good amount of them available on the platform, and not only can those created capabilities be used but also cloned and edited to fit various needs. Since Snips is open source, everybody can use it to create a smart device using a Raspberry.

There is another open-source virtual assistant called Mycroft [15] that is very similar to Snips in terms of features. One of the selling points of this virtual assistant is that Mycroft

is modular and users can change the core components, for example, the TTS or STT engines, without changing the other components.

## 3.2. Messaging Platforms

---

Messaging platforms are often used to exchange messages between users but nowadays, since the communication between users and chatbots has grown, they are used to connect users with chatbots. Since communication between users and chatbots is growing, the platforms created some tools available that help publishing chatbots on the platform.

Facebook Messenger [16] is a messaging platform created by Facebook and has become one of the most used as of today. The Facebook Messenger platform has a series of tools and APIs available. The APIs and tools available are mainly used to integrate a chatbot with the platform but there are many other features available too, for example, processing natural language, accepting payments or monitoring the chatbots' performance.

Telegram [17] is another messaging platform that allows integration between the platform and chatbots. In order to integrate chatbots, Telegram has an API called Bot API that allows creating a special account that is used by the chatbots.

WhatsApp [18] is another messaging platform, and is one of the most used. Despite that, they only recently announced the WhatsApp business API that will allow the connection between a chatbot and the application. This API is not available for everyone, but it is possible to request a preview of the API.

There are many messaging platforms available and the traditional Short Message Services (SMS) applications have evolved and became Rich Communication Services (RCS) that are, in terms of features, very close to the messaging platforms and one of those features is the implementation of APIs that allow the communication between chatbots and users. One of the key differences is that RCS applications can use a SIM card to send messages between users while most of the messaging platforms only work while using the internet. One thing to notice is that messaging platforms like Facebook Messenger started to offer the possibility to associate the SIM card in order to be able to send messages as a traditional

SMS application would. This means that messaging platforms are aiming to become more like RCS applications by offering integration with traditional communication services.

### 3.3. Chatbots Platforms

---

Messaging platforms created tools and APIs that allowed the publication of chatbots on the platforms natively and that lead to the appearance of chatbots platforms.

Chatbots platforms essentially provide users an easy way to develop and publish chatbots on the messaging platforms by implementing, internally, the tools and APIs that the messaging platforms provide. One of those platforms that facilitate the creation of chatbots is GupShup [19]. GupShup is one of the top platforms available on the market that has a series of tools to create and publish chatbots on multiple channels. The platform allows users to create bots using an online Integrated Development Environment (IDE) or using a visual interface that is designed to be used by non-programmers. Afterward, the same chatbot can be published in many different channels. GupShup also has another platform that enables communication between chatbots and is currently the only platform that has that functionality.

There are also platforms, such as DialogFlow [20], in which even though users can create chatbots on the platform the selling point is the NLP system. DialogFlow is often used in other platforms to perform the NLP process. The platform itself lets the user create intents and entities that are the core functionalities to create an *agent* (that can be interpreted as a chatbot) and deploy that agent in multiple channels, such as Facebook Messenger. DialogFlow is optimized for the creation of Google Assistant actions.

Wit Software also has a platform to create chatbots that is currently under development. The platform, that is called Wit Bot Engine, allows users to create bots and create conversational flows using BPMN diagrams. The platform is easy to use because it is designed to be used by people that do not have coding skills. Despite that, users can still code tasks inside the conversation flow using NodeJS and it allows users to extend the chatbot behavior. Since this platform is directly related to the internship project, a more detailed description of its features will be given in section 6.1.



## 3.4. Chatbots Languages

---

Chatbots behavior is, most of the time, described using an AIML file that is parsed by an AIML interpreter/engine. This AIML interpreter/engine will also be responsible for, upon receiving user input, replying using the rules described in the AIML file. There are several implementations of this AIML interpreter in different languages which allows the use of AIML in many programming languages such as Java, PHP, etc. This brief explanation overviews how the rules described on the AIML file provide the chatbot with a behavior.

AIML [21] is a standard, which means that is used by many of the platforms described in the subsection prior to this one; nevertheless, it is important to explore alternatives. During the research for AIML alternatives several alternatives that share many features with AIML were identified that, at the same time, have their own strengths that are what differentiate them from AIML. Choosing a language to define the behavior of a chatbot will depend on the complexity of the chatbot that the users want to develop and how comfortable they are using that language.

ChatScript [22] is one of the most feature-complete alternatives to AIML. The ChatScript syntax is not based on XML, which means it is very different from the syntax used on AIML. Instead of having tags it uses symbols on the start of each line that are used to describe triggers and responses to that triggers. The rules are described in files with the “.top” extension that stands for “topic” because the chatbot behavior can be split into multiple files which each could represent a different topic. The ChatScript interpreter/engine was developed in C++ but it is easy to run a server and send an HTTP request to the engine in order to get an answer for given user input.

Similar to ChatScript, there is also RiveScript that is described as a simple scripting language that has a friendly and easy to use the syntax [23]. RiveScript syntax is also very different from AIML but is a much more simple and easier to use than the one that ChatScript uses, but it is not as feature-complete as ChatScript is. Similar to ChatScript, RiveScript uses symbols at the beginning of each line to represent the triggers and the responses for that triggers. RiveScript can be used in multiple programming languages such as GO, Java, JavaScript, PERL or Python because there are interpreters written in those languages. RiveScript aims to be a simple, powerful and flexible way to create chatbot conversations

and that is why it is a good alternative for beginners. RiveScript was also an inspiration for Superscript and they have many similarities especially at a syntax level.

Intelligence Markup Language (SIML) is one of the best alternatives for someone that is looking for a replacement of AIML but does not have time to learn new syntax (in terms of syntax is very similar to AIML). One of the downsides of using SIML is that it was designed to be used using C# which means there is no interpreter available for other languages. Even though it shares some similarities with AIML there are some features, such as the possibility to have multiple patterns on the same category that are considered to be an improvement over AIML.

Similar to ChatScript and RiveScript, there is BotML. BotML is described as a simple, modern and highly reusable syntax for writing powerful chatbots [24]. At a syntax level, it is easy to understand because it uses symbols at the beginning of each line instead of complex tag elements. One key difference between BotML and the other alternatives is that version 2.0 cannot be used for commercial purposes without a commercial license which means that there is an associated price.

Any of these chatbot languages can use wildcards and elements to capture and save entities or information from user input. It is also possible, in most of these chatbot languages, to extend the chatbot behavior by creating functions that can be injected or called from the files that define the bot rules and it allows a much more controlled and powerful chatbot behavior. This will allow, for instance, to call a web service to tell the user the weather for tomorrow.

## 4. Requirements and Functionalities

---

Given the project's premises, it is possible to define a series of use cases that would be either necessary or nice to have in order to reach the main goal of this project, that is the communication between chatbots.

A list of several use cases was created which are shown in Table 1. The table shows the use cases and an evaluation of the necessity of each, meaning that some of the use cases could and have been left behind in order to have the necessary features working. In this case a "should have" feature does not necessarily mean that the project main objective will not be achieved, because in order to achieve the main objective of this project the core feature is the creation of a communication channel that would allow the communication between two chatbots on the platform and achieving that will guarantee that the main objective will be achieved. The "should have" features will define what should be done in order to create a better user experience. The "should have" features were suggested by the development team because during the research phase the team saw what was already on the market and collected important data. This data was used to understand what could be done, what could be improved and how could this product be different from the already existing ones.

*Table 1 - Project Use Cases and priorities*

Use Case	Prioritization
<b>As a user, I want my bot to be able to communicate with bots created on Wit Bot Engine platform (Federation Channel)</b>	Must have
<b>As a user, I want to be able to federate a bot from Wit Bot Engine</b>	Must have
<b>As a user, I want to be able to add Google Assistant on the skill mapping</b>	Should have
<b>As a user, I want to be able to federate bots from Facebook Messenger</b>	Nice to Have

<b>As a user, I want to be able to create mappings between a skill and a bot</b>	Must have
<b>As a user, I want to be able to list all chatbots that can be used for the skill mapping</b>	Must have
<b>As a user, I want to be able to list all start events from a bot to be used on the skill mapping</b>	Must have
<b>As a user, I want to be able to edit a skill mapping</b>	Should have
<b>As a user, I want to be able to remove a skill mapping</b>	Must have
<b>As a user, I will not be able to configure a stop word so the chatbot users can easily end the conversation with a delegated bot</b>	Must have
<b>As a user, I want to be able to enable and disable the federation channel for each bot</b>	Must have
<b>As a user, I want to be able to configure federated bots' permissions</b>	Should have
<b>As a user, I want to be able to enable and disable the federation</b>	Should have
<b>As a user, I want to be able to configure a threshold for the accuracy of Wit Bot Engine bots.</b>	Should have
<b>As a user, I want to be able to train skills for bots from external channels</b>	Must have
<b>As a user, I want to be able to list all the mappings</b>	Must have

One of the key features that is needed, in order to fulfill all the use cases mentioned above, is the creation of a communication channel that allows the communication between chatbots that are hosted on the Wit Bot Engine platform. Many of the features are for the skill mapping represent the connection between a delegated bot in a channel and a skill. Skills for the bots hosted on the Wit Bot Engine are considered to be the start events and for the other channels is an event with trained utterances. The uses cases mentioned in Table 1 are for the back office, and hence used by users whose main role is to manage and create

chatbots on the platform. There are also other features in which the main focus is to slightly change the experience from the perspective of a user that is talking with a chatbot.

## 5. Planning and Methodology

The project was planned in six different phases as it can be seen in Figure 3. The first phase of this project was where the necessary research is done. The research's main objective was to search and test different tools and APIs that are somehow related to the project and later could be used on the project. It was also helpful to gain some knowledge about some of the key concepts related to the project.

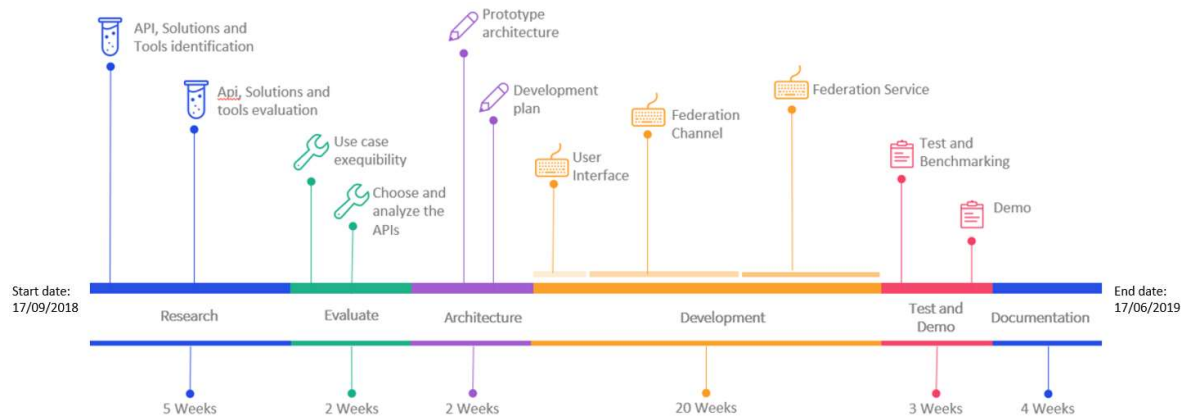


Figure 3 - Project Schedule diagram

The second phase was be the evaluation phase, that is directly related to the research phase. The evaluation phase was when, after testing some APIs and tools, it was possible to decide which APIs or tools would be used on the project. After deciding which tools and APIs would be used on the project, it was important to create an architecture for the project and create a plan for the development phase.

The development phase is where most of the internship time was be spent. The user interface was be the first thing to be implemented, and only after the interface was finished the features were developed. The first feature to be implemented was the federation channel, that is the key part of the project and will be responsible for opening the possibility to connect two bots available on the Wit Bot Engine. After having a channel where the bots can exchange messages, the skill mapping feature was be implemented. The skill mapping allows users to map different skills to bots, making it easier to decide which bot is the best to answer the user's question. Another important feature is the connectors, that are responsible for telling where the bot is hosted and how it is possible to talk with it.

When the development phase was finished, a demo of the project was created in order to show the features completed on the development phase. Some tests and benchmarks were also performed to see if the project does what it is supposed to do and how well it does.

The last phase was the documentation phase where the unstructured information created during all the other phases was revised and structured. This data is relevant to the report, which means during this phase one of the priorities was to add this relevant information to the final report.

The methodology used during the internship was an adaptation of SCRUM [25]. SCRUM is an agile process framework used for product development that is iterative and incremental.

SCRUM has essentially three roles:

- Product Owner – Represents the stakeholders and is responsible to capture the customers' expectations, preferences, and aversions;
- Development Team – Responsible to develop a releasable product that increments every sprint.;
- SCRUM Master – Responsible to ensure that the SCRUM framework is followed and is accountable to remove any impediments to the ability of the team to deliver the product goals and deliverables

Since this was an internship project the development team was composed by only one person that was me and the role of SCRUM master and product owner was fulfilled by the tutor that supervised my work at Wit.

The SCRUM workflow starts by defining the product backlog that is a breakdown of the work that needs to be done and contains an ordered list of the product requirements. At the beginning of each sprint, there is a sprint planning where is discussed the scope of the work of the sprint that is going to start. The sprint should last no more than a month and at the end, a sprint review is done where the work done during the sprint is reviewed and work can be shown to the stakeholders. During the sprint there are daily SCRUMs, that should not last more than fifteen minutes, where team members share what they did, what they are planning to do and if there are any impediments.

The methodology used is an adaptation from SCRUM because:

- The daily SCRUM was not done every day, but done once a week when possible;

- The development team had only one developer because it is an internship and the development had no interference with the Wit Bot Engine team;
- The product requirements were defined by the development team with the validation of the SCRUM Master/Product Owner;
- The role of SCRUM master and product owner was fulfilled by the same person.



## 6. Technologies and Tools

---

One of the internship's goals is to create a project that can be integrated with the already existing Wit Bot Engine platform. Given that there are several tools and technologies that will be used to keep this integration as simple as possible.

In this section, an overview of the Wit Bot engine platform will be given as well as some information about the technologies used.

### 6.1. Wit Bot Engine

---

The Wit Bot Engine is a white label platform, which means that it is a generic product that can later be labeled in order to be used by any brand, organization or group. The platform is designed to be installed on the company's premises making it more secure and, more importantly, making it more private because all the data, especially the conversation context data that usually contains sensitive data, will be stored on the company premises.

One of the key objectives of this internship is to integrate the project with the already existing Wit Bot Engine [26]. Wit Bot Engine is a platform that, allows users to create and manage chatbots but, since the integration is one of the objectives, a more detailed explanation of the platform's features will be given on this topic to give some introduction to features that could also be related to the internship.

The most basic feature the platform has to offer is the possibility of creating a chatbot by simply clicking on the "add new bot" button, and is present on the first page if the user is already logged in. Every bot created is associated with an entity, and entities can be organizations, brands or a group.

When the user clicks on the button a new dialog appears where the user can fill the details about the bot that he wants to create, for instance, the name of the bot, a time zone, and an image. This creation dialog can be seen in Figure 4.

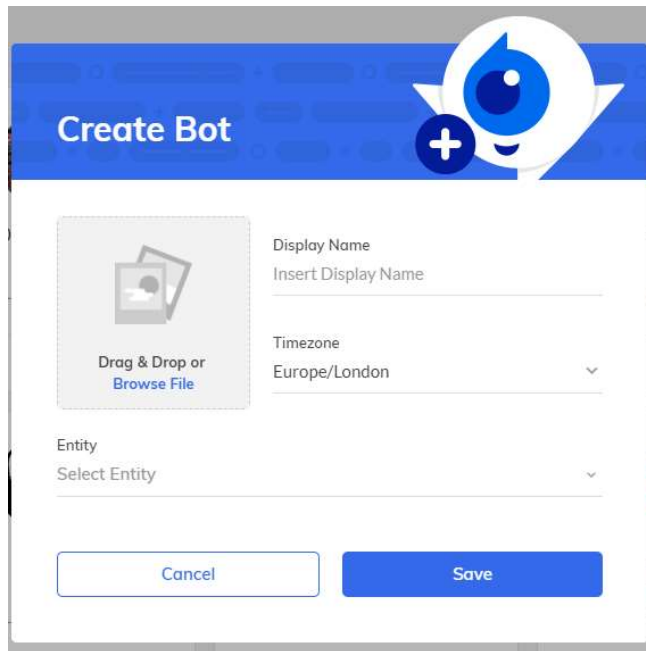


Figure 4 - Bot creation dialog

When the user saves and closes the dialog, the bot will be created and will be added to the list of bots available on the platform, which is usually shown on the first page if the user is already logged in but, at this point, the chatbot created is only an empty “shell” that can be edited to become a real chatbot.

In order to create the chatbot conversation flow, the user can click on the bot and go to the builder shown in Figure 5.

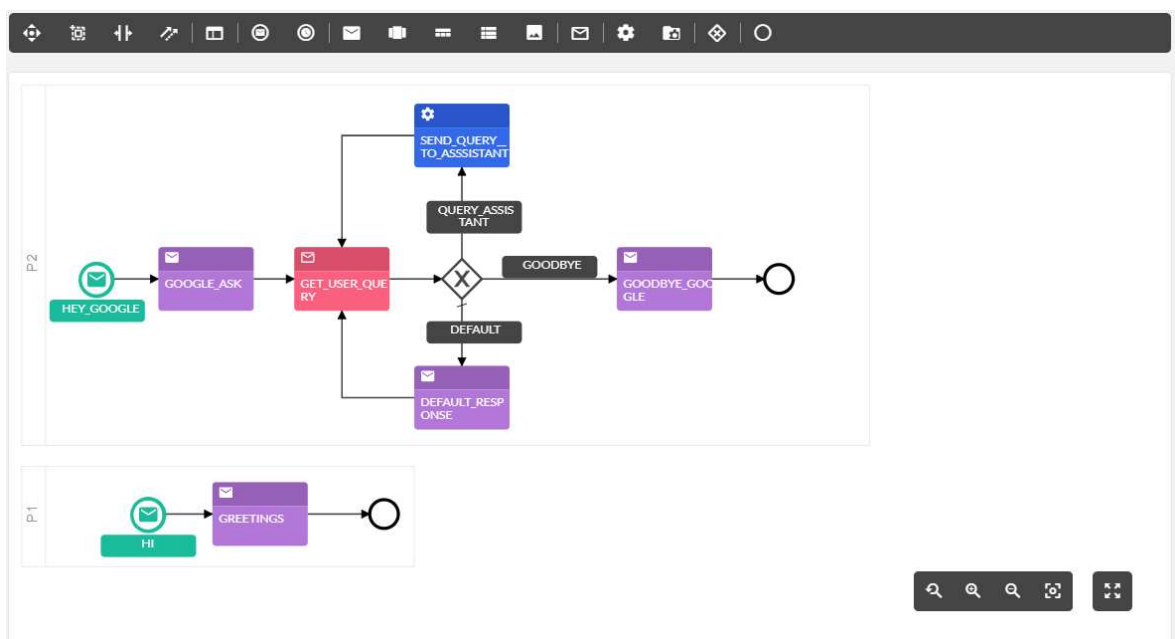














Figure 5 - Bot conversation builder

The conversation flow is a series of components that describe what the chatbot can do and when it should be done regarding the current conversation context. The conversation flow is designed using a business process model notation (BPMN) that has the different components that can be used to define chatbot behavior. The components available on the platform can be seen in Figure 6 with a brief description for each.

Activity	Description
 Start Event	Defines a starting point for the process.
 Start Timer Event	Schedules a timer event.
 Send Text Task	Sends a text message.
 Carousel Send Task	Sends a carousel message.
 Quick Replies Send Task	Sends a quick replies message.
 Buttons Send Task	Sends a buttons message.
 Image Send Task	Sends a message containing an image.
 Receive Task	Waits for user input.
 Exclusive Gateway	Decision point to adjust the path of a flow based on expression conditions.
 Sequence Flow	Connection between 2 activities.
 Service Task	Executes custom or predefined code.
 End Event	Ends the current process instance.

*Figure 6 - Bot builder components*

Using these components and connecting them will allow users to create complex conversations flows. This conversation flows are easy to set up because they are designed to avoid coding but, if the user knows how to code in NodeJS, there is a component called “service task” where the user can write NodeJS code. This component is important because it allows, to a certain degree of freedom, the creation of more complex behaviors and is especially useful if the bot needs to access an API to get and then parse information. The service task has functions related to the bots that can be used to send a message to the user, get captured user input, get user ids, get the channel the user is using to talk with the bot and even to log important information and errors.

The platform is also multi-channel, which means that when a user creates a chatbot it can be published in multiple channels. This approach will save users time because the only

thing that the user needs to do is configure the channel properties and activate the channels that he wants, and the exact same bot will be available on the channels he chose.

When a user changes the chatbot conversation flow the platform will not immediately change the version of the bot that is published on the channels, so that the user can edit the flow without compromising the published version of the bot. In order to publish a change, the user needs to approve and clearly tell the platform that he wants to publish the current version with all those changes. This will also allow users to rollback to previous versions of a chatbot if needed and, for example, delay the deployment of the new version to avoid changes to the bot flow when there are too many users using the chatbot.

On the platform, it is also possible to configure external NLP systems (for example, DialogFlow) so the user can have access to a more robust NLP system that can improve the success rate of the system, which means the users that will use the chatbot will have a better overall experience. Besides that, Wit Software also has a proprietary NLP system, which is even easier to use because it requires fewer steps to configure.

One of the key features is that the user can add multiple languages; so, with only one flow, it is possible to create a bot that can answer in different languages. If a user sends a bot a message in Portuguese the bot can answer in Portuguese, but if another user sends a message in English the bot can also answer in English. It is also possible to customize each chatbot to have “personas”. Personas are essentially used to, for the same flow path, give personalized answers based on the target audience. Giving a real example: if a user is underage and sends a message saying “hello” to the chatbot, the message sent back by the chatbot could simply be a message saying “yo” and, for an older person, the same user input could result in a different output like a message saying “hi”.

Chatbots cannot be perfect and that means they can fail, and the platform keeps track of misses to enable checking if human intervention is necessary. Before human intervention, it is necessary to fail more than once in order to reach this step because, before there is the default response and the steer back response. The default response is the response triggered when it is not possible to get an answer; the steer back response is similar to the default answer but redirects the conversation to a given flow path. This is an important feature because if there are too many misses, chances are that the user is asking something that the bot he is talking with cannot answer, and it is better to handle the conversation over to a human in order to avoid frustrations.

The platform itself is well documented and users have access to information about all the different features that are already available on the platform, and a series of video tutorials on how to create a chatbot on the platform that also show how to use the different components. It also has some NodeJS code snippets that teach users how to use the available functions and variables for the service task component. This documentation is available on the help section of the platform so it can be used by the companies that have access to an environment where the platform is deployed.

There are other features, like the creation of new entities and the management of the users associated with an entity, that are only available for specific roles. The roles available are:

- **Super Admin:** has full access to the system;
- **Entity Admin:** cannot manage entities, but can see and do everything else, limited to the assigned entities;
- **Bot Admin:** cannot manage entities or entity users, but can see and do everything else limited to the entities that he/she is assigned;
- **Bot Designer:** can see the bot's list for the entities assigned and can use the Wit Bot Builder;
- **Conversation Designer:** can see the bot's list for the entities assigned and can use the pages on the Conversation Management;
- **Knowledge Base Designer:** can enable the AI Knowledge Base capability on a bot and manage all the related information.

## 6.2. Angular

---

Angular [27] is a framework that is commonly used to create web applications. There are many advantages that this framework brings to web development. One of those advantages is the fact that it is possible to create components that can bring a modular approach to the project. This component-based system helps to reduce the code that ultimately can lead to an easier to manage the project. Angular is also dynamic, which means that variables can be referenced on the views and when those variables change the views are

notified. This is important because unlike other frameworks like jQuery, it is not necessary to access the document object model (DOM) directly in order to edit the data shown on the views. It is also important to mention that Angular views are written in HTML, that is a standard language for web development that is easy to learn. Angular also uses typescript that is a superset of JavaScript that brings static typing and object-oriented programming to the developers that are familiar with JavaScript.

## 6.3. Java/Spring Boot

---

Java is an object-oriented language that is well known because is one of the most used programming languages. The simplicity that makes it easy to learn and the age that brings maturity to the language are the key ingredients to its success.

The Wit Bot Engine platform services are written in Java using the Spring Boot framework. The Spring Boot [28] framework is a framework used to facilitate the creation of standalone and production-grade Spring applications. Spring Boot is used because automatically configures spring and third-party libraries when possible, embeds Tomcat and simplifies the build dependencies.

## 6.4. Eureka

---

The Wit Bot Engine platform is based on a microservices architecture, which means the platform has several services that communicate with each other so that it is possible to scale up or scale down each service individually. Since there are several services this means that it is necessary to have a solution that can list all the services in a single spot so that all the other services can register themselves in this list and, at the same time, search for all the other services to make it possible to achieve communication between two services. This is where Eureka can be useful. Eureka is a project developed by Netflix that is responsible to provide all the necessary tools to create a project that has multiple services using a service registry mechanism. Eureka not only is responsible to provide the service registry capability,

but it is also responsible to provide load balancing capabilities by implementing the necessary mechanisms to redirect the requests when there is more than one instance for the same service.

The Wit Bot Engine has a service called “Service Discovery” that implements the Eureka service and is used as a bridge between all the services. This service is used in most of the other services in order to be able to find and send requests between the different services.

## 6.5. gRPC

---

gRPC is a Remote Procedure Calls (RPC) framework [29], developed by Google, that is often used to create a connection between multiple services. The Wit Bot Engine Platform has a microservices architecture and in order to connect all these services gRPC is used. The gRPC framework is used because it allows bi-directional streaming, works across multiple languages and is scalable. The platform has scalability in mind which mean that a framework like gRPC is important and, although most of the services are spring boot applications, there is at least one service that is written in nodeJS and the gRPC framework can be used to connect a nodeJS service with a java service easily. Another important feature is that gRPC is cross-platform which allows solutions, that use gRPC, to be deployed in multiple platforms.

## 7. Development

---

This section will contemplate the most important subjects about the development process. That includes not only the decisions made before the creation of the prototype but also the actual process of prototyping the solution and implementations of it. Some instructions on how to configure a federation on the Wit Bot Engine by showing how to create the necessary chatbots and which settings should be configured to use the project's features.

### 7.1. Decisions Made

---

Projects tend to have a good amount of ideas at the beginning that needs to be explored so it is possible to clearly identify what should be carried into the prototype development. On that note, some decisions were made in order to keep the project feasible during the internship's duration (nine months) and avoid features that could not be implemented due to technological limitations. The most important decisions are documented, mainly to clarify why certain features were not implemented.

One of the requirements for this project was to make it possible to federate bots from other platforms, such as Facebook Messenger. In the research phase, a non-official API [30] that can simulate a conversation done on the Facebook Messenger platform was found. Since the official API did not allow communicating with a bot and the API found that would allow it is not official, it was decided that this feature would not be implemented.

Some other proofs of concept were implemented during the research phase in order to test the possibility to integrate the Google Assistant and Alexa on the project. Doing so made it possible to conclude that Alexa (even though Amazon released a service to communicate with Alexa) would not be a good fit to the project because the service used to communicate with Alexa only accepts audio inputs, and the communication with chatbots are usually done using text inputs. Research was done to find a solution for this problem and found a project called Virtual Alexa [31]; with virtual Alexa, sending text inputs to talk with Alexa is



possible, but even though it solves the input problem it brings new problems to the table. One of the biggest problems is that keeping multiple contexts is not feasible unless there is a virtual device for every user. Another problem associated with Virtual Alexa is that in order to make it work using text inputs, virtual Alexa converts the text into an audio file and sends it to the Alexa voice service; since Alexa voice services are trained using human voice inputs, the results for audio inputs generated using a TTS engine are far worse. For those two reasons, it was decided that this approach had so many problems that the idea of being able to federate Alexa was dismissed.

Since Alexa was not an option, a decision was made to keep the Google Assistant on the project, not only because it is feasible and interesting in terms of the projects overall value, but also because Google Assistant and Alexa are very similar in terms features. The objective of Google Assistant is mainly to be used when all the other chatbots cannot answer, although this behavior can be configured.

Since the platform Wit Bot Engine is a proprietary system created on Wit, it is possible to use the bots created on the platform on the federation. Since it is possible, it was decided that this feature was to be included.

It was also decided that, in order to simplify the process of conversation delegation, it an extremely advanced machine learning algorithm should not be implemented. Regarding which bot the conversation should be delegated to, it was decided using a direct mapping between the intents or start events and the delegation bots. With this mapping, the NLP system can be used to help with the decision. In cases in which two bots have the same type of intents and they are created on Wit Bot Engine, the confidence level will impact the decision, but on all the other channels where those type of metrics do not exist, the first bot found for that intent will be the one used.

The fact that the conversation is delegated to another bot lead to a problem where the user could be trapped in a conversation that he did not want to, so it was decided to create a stop word that gives the user more control on the conversation and gives him the possibility of quitting the conversation with a bot at any given time.

Regarding the technologies that are being used on the project, they are not intended to change during the development process. This is due to the fact that they are being picked because they are already being used by the Wit Bot Engine project and using the same technologies, and their usage would thus make it easier to integrate both projects.

## 7.2. Architecture

The project architecture, which can be seen in Figure 7, was based on the already existing Wit Bot Engine's architecture. Since this project involves several components that must integrate with the platform, some common components are shown in order to demonstrate where the integration between the platform and the federation project is performed. Since some of the existing components are directly and indirectly connected to the internship project, they suffered some changes to, at least, have the necessary information about the data that flows between them that are related to the federation itself.

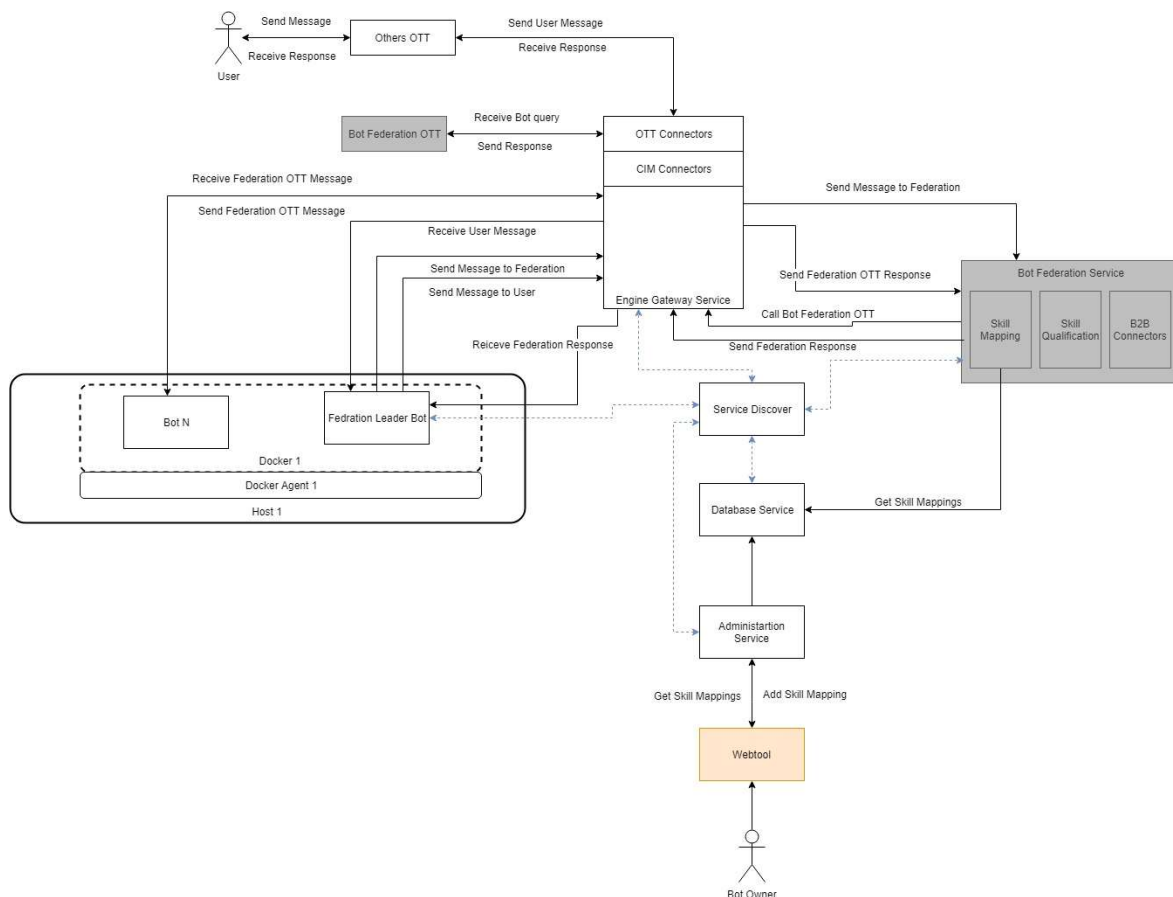


Figure 7 - Architecture diagram

The Federation channel is the communication channel responsible for creating the connection between two bots that are available on the Wit Bot Engine platform. This channel is similar to the ones that already exist but, instead of the message sender being a user, the message sender is another bot. This channel's main purpose is to open a door to allow the communication between bots that were developed on the Wit Bot Engine platform.

The bot federation service is a Spring Boot service that will be called when the federation path is reached. This path can be reached when a user message is sent to a bot, but the flow cannot find an answer for that user input. This service will be responsible for handling the user message and delegating that message to a bot that was previously mapped to handle that message. The web tool is a component that already exists for the Wit Bot Engine, and it is an Angular project that can be deployed on the web with all the necessary features to create and publish a chatbot. In order to have a place where the user can manage and create federations, it will be necessary to change the web tool. The change on the web tool will allow the users to create and manage federations with an interface that is familiar to the user that already uses the Wit Bot Engine platform.

When a message is sent by a user, a webhook available on the gateway service will capture the user message. At this point important information, such as the user unique identifier, is cached. The gateway service will redirect the message to the “Federation Leader Bot” that can be seen in Figure 7. If this bot can answer the message a response is sent back to the gateway service that will use the information cached before in order to be able to redirect the message back to the user. If the “Federation Leader Bot” cannot answer to the user input, the message will be redirected to the federation Service where the process of selection will be done. From the federation service the message is redirected, using the federation channel (Bot Federation OTT) to the bot selected to answer the user input. In order to retrieve the message to the user, the message is sent back to the bot that is talking with the user in order to be able to reach back the user. This is a simple explanation of the user message flow, from the point where the user sends a message until he receives a response. There are several other services that intervene in this flow but that are not particularly important for this internship project.

## 7.3. Implementation

---

The development of this project is focused on the creation of a prototype that explores the possibility of communication between chatbots, especially chatbots that are hosted on the Wit Bot Engine platform. It is not an objective of this project to create a product ready for the market, but instead one that acts as a feature proof for the concept.

The first step of this project is the creation of the interfaces, in order to have a solid base where it is possible to connect the service so the user can easily access the project available features from the platform website. There are two important goals that should be achieved with this interface: one is to use the already existing platform's interfaces design principles; and the other is that it must be as user-friendly as possible.

From the idealization and implementation of this project two types of bots were defined that represent the bot role in a federation:

- Leader/Master bots – Responsible for handling the user input and manage the communication with the other bots;
- Slave bots – Bots that can be reached by the leader bots using the federation communication channel or other external channels such as Google Assistant.

The leader bot is responsible to talk with the slave bots, while a slave bot acts as an asset that can be used by the leader so that it is possible to send messages from the leader to the slave. These two roles are not restrictive because if the federation channel is configured on what is considered a leader bot, this means that he can be also considered a slave bot because it is possible to communicate with this bot through the federation communication channel. From the platform's point of view, every chatbot starts as an "empty shell" that is the base for the creation of a new chatbot, which means that initially there is no difference between a leader bot and a slave bot. Having this in consideration a bot role in a federation is only defined by how the bot is later configured through the Backoffice platform. These roles are not mandatory, which means that bots without federation roles could also exist in the platform, meaning that they cannot communicate with other bots or being used by a leader bot.

From the user that talks with the bot perspective, all the communication is done with a leader bot because he is responsible to manage and handle the conversation between the user and the slave bots. It is important to keep the user aware of what is happening so that they always have information about the bot they are talking with. This information is displayed in two different ways:

- Text – A message warning the user about the bot that is going to handle the conversation from that point on, as well as the information about the delegation ending;

- Personas – Used for Channels, like Facebook Messenger, that allow the usage of personas which change the name and the avatar of the message sender, on the chat, to simulate a message sent from another person.

## 7.4. Backoffice

---

The Backoffice is an online platform developed in Angular that communicates with the administration service and acts as a gateway between the Backoffice and the other services.

The Backoffice is mostly used to create, configure and manage the bots because every bot starts as an “empty shell” that later, after some configurations, acquires the desired behavior. This project requires several configurations to be done by the person who designs the chatbots which means that these configurations need to be available on the Backoffice. To be able to configure a federation there are at least two new interfaces on the Backoffice and several other interfaces that need to be changed.

The most important interface that is going to be used to configure the federation is the interface where it is possible to create all the necessary mappings between the leader bot and the slave bots. This interface can be reached by clicking on the Bot federation option that is on the side panel that can be seen in Figure 8. As it can also be seen on the table in Figure 8 there are already two mappings configured for the Master Bot federation. To be able to

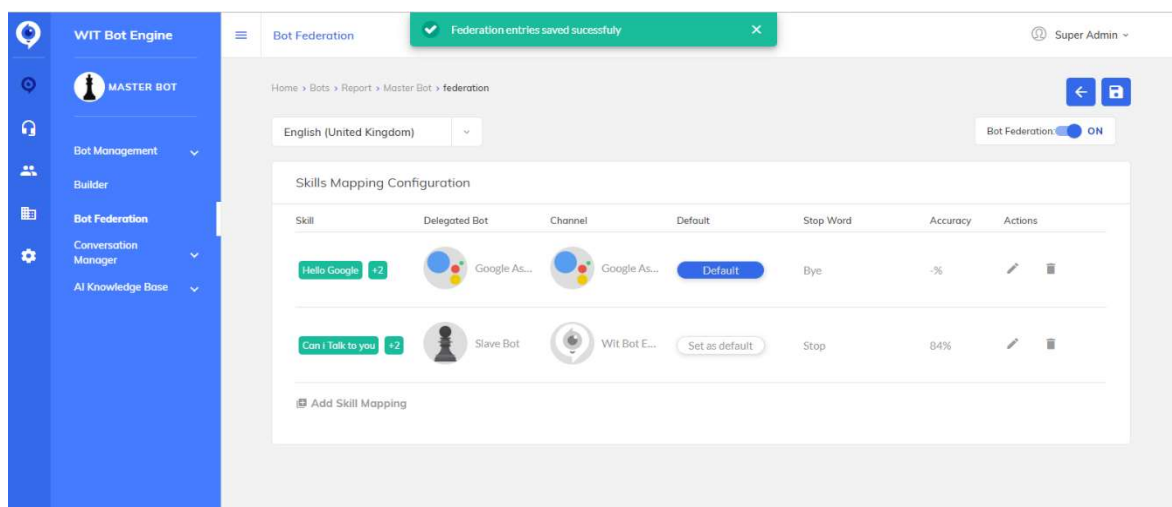


Figure 8 - Skill Mapping Interface

configure and create a federation, the Bot federation needs to be activated by clicking on the toggle button that is near the top right corner of the mappings table.

In this table, it is possible to see what are the skills that are mapped, the bot that has such skills, what is the channel that can be used to communicate with him, the stop word, the accuracy and if it is going to be the default mapping. The default mapping is used to force the usage of a certain mapping if the algorithm cannot find a chatbot that can answer the user's input. In the case shown in Figure 8, the Google Assistant is going to be used when the algorithm fails to find a better mapping. Regarding the accuracy, this value is used to mainly reject a certain mapping if the level of confidence of the NLP system is lower than the mapping configured accuracy. On the mapping table, it is also possible to perform the edit and delete mapping actions. If a user wants to create a new skill mapping, there is a button that is always on the bottom of the table. When the user clicks this button, a new dialog appears.

The dialog that can be seen in Figure 9 is where the user starts the creation of a new skill mapping by selecting the channel that needs to be used to communicate with the slave bot. In this case, there are only two options available:

- Wit Bot Engine - Channel used to communicate with other bots available on the Wit Bot Engine platform;
- Google Assistant - Channel used to communicate with the Google Assistant.

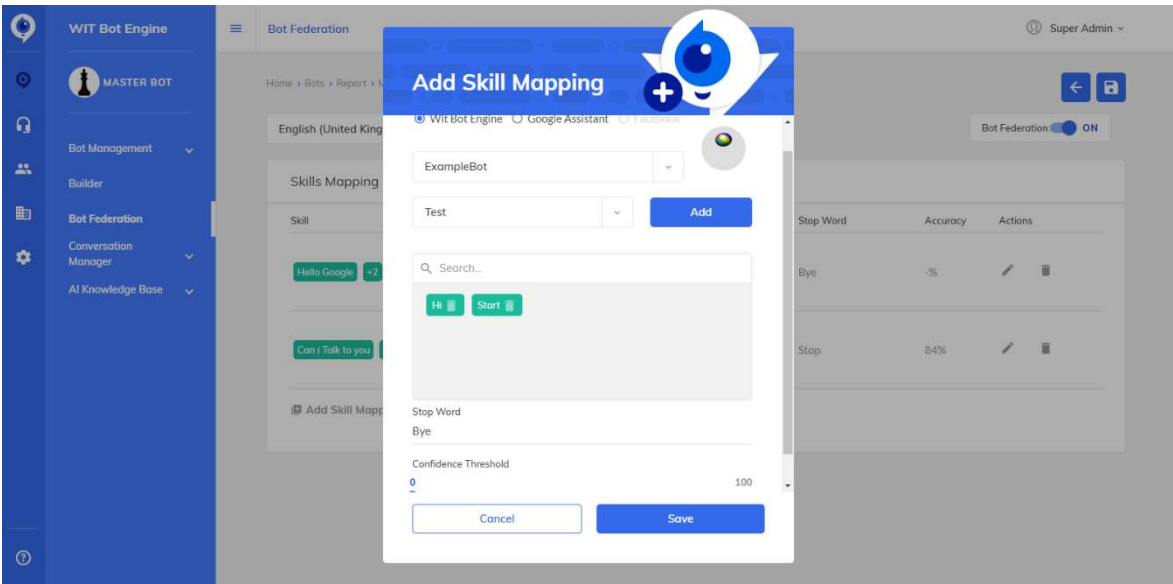


Figure 9 - Create/Edit Skill Mapping Dialog

Based on the channel selected the configuration will be different. If the user selects the Wit Bot Engine channel the properties that can be configured are the ones shown in Figure 9. The user can select the bot that he wants to map, add all the necessary skills, configure the stop word and select the confidence threshold that is the same as the accuracy on the table. To make it easy to find a specific skill when there are too many available for one bot, it is possible to search and filter from the list of available skills. On the other side, if the user selects the Google Assistant channel, since there is only one bot that is the Google Assistant itself, the interface will be a little different. The Google Assistant configuration interface will ask for the refresh token because Google Assistant requires authentication and the skills will be given manually because there are no skills configured for external channels.

The interfaces seen in Figure 8 and Figure 9 are interfaces used to configure a

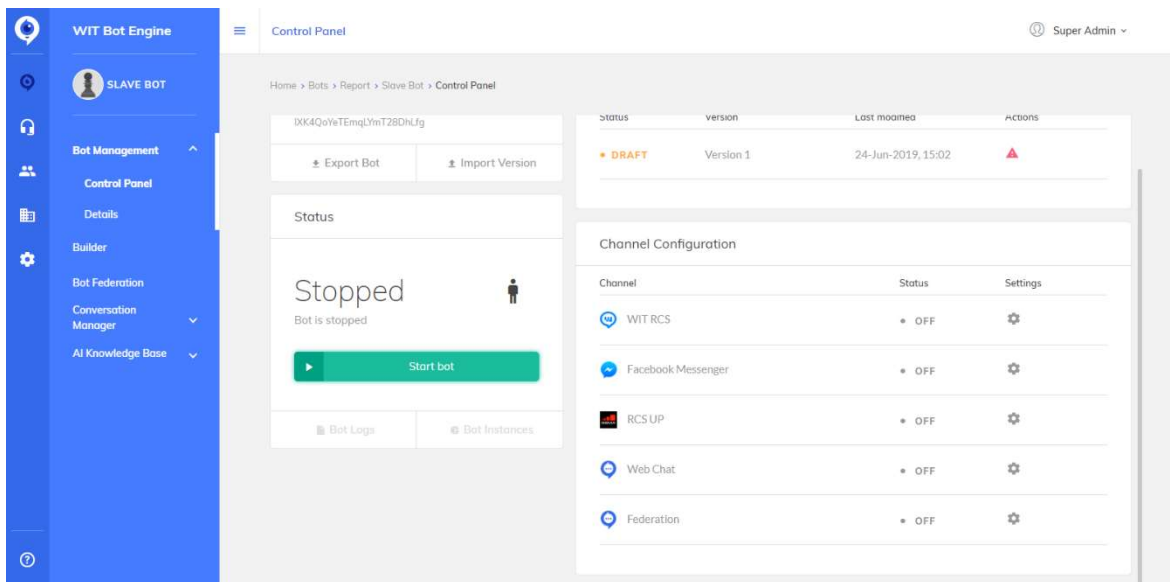


Figure 10 - Bot Control Panel

federation which means that these interfaces should be used to configure the master bot. If the Backoffice user wants a bot from Wit Bot Engine to be used in a federation as a slave bot he needs to access a whole new interface that is used to configure the federation channel that is the channel used to communicate with bots available on the Wit Bot Engine. To access the interface the user needs to select the slave bot, and in the side panel using the control panel option that is a sub-option inside the bot management option, as it can be seen in Figure 10. When the user opens this option, a list of channels is available to be configured. In the list, there is a new channel called 'Federation'. To configure the federation channel the user needs to click on the icon below the settings column, and a new interface will appear. The

interface is shown in Figure 11 and there are several aspects that can be configured on the slave bot.

On the left side of the interface, the user can configure the slave’s availability. This configuration will tell if the slave is available to be used by all the other bots or by none of the other bots. It is also possible to create a custom list of the bots that can and the bot that cannot talk with this slave with a whitelist and a blacklist respectively, which is the option

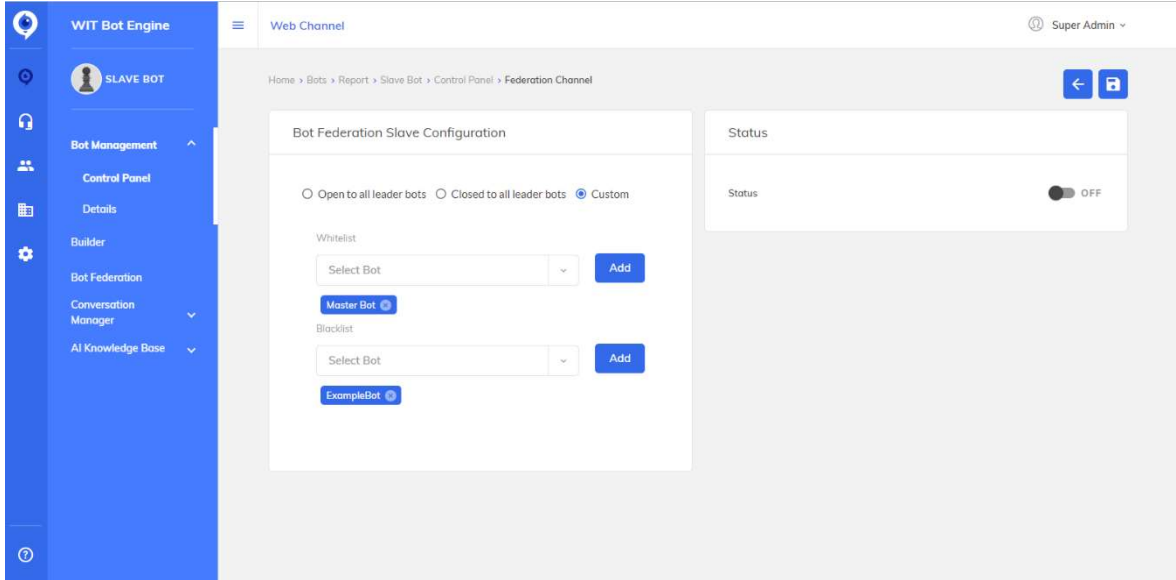


Figure 11 - Federation Channel Configuration Interface

that was selected in Figure 11. On the right side on the status panel it is possible to enable and disable the channel that opens and closes the communication channel itself.

If the user does all these configurations, he can create a federation and have a master bot that can delegate the user input when it is not possible to answer, giving the user the best answer possible.

## 7.5. Backend Services

---

The wit bot engine backend is the combination of all the services. To better understand the role of each service on the federation project, an explanation about what is the role of the service and what were the necessary changes to accommodate the federation features will be given on the following subsections. During the internship, not all services were used or



changed which means that there are services seen in Figure 7 that are not going to be detailed in this section.

### **7.5.1. Administration Service**

---

The Wit Bot engine platform uses a microservices architecture which means there are several services that run in the background used to perform all the actions done on the Backoffice. To interact with all these services from the back office there is a specific service called “Administration Service” that acts as a gateway between the platform and the other services. Since there are new features available on the Backoffice this reflects in features that need to be implemented on the administration service in order to interact with the other services to perform the necessary actions.

Since the communication between the Backoffice and the administration service is done using Representational State Transfer (REST) calls, several controllers were changed or created to accommodate all these new changes. The new features created on the administration service are also role protected so that only certain roles can perform those actions. For example, in order to be able to create a new bot, only the role of bot designer and above can perform such action. In order to be able to choose which roles can configure a federation, only the roles with the necessary permissions can actually use the new Backoffice interface used to configure a federation.

### **7.5.2. Runtime Service**

---

The runtime service is essentially where the generic flow path of the chatbots’ behavior is defined. As mentioned before every chatbot starts as an “empty shell” that has its own runtime instance that, when given the right properties such as the bot name, can acquire the desired behavior previously configured in the Backoffice. The bot has several steps to seek for an answer that, in its most simple form, will start at a point where he tries to find the answer in the configured core knowledge defined by the bot designer on the conversation flow using the Backoffice. From that point on the bot will try to seek the answer by other

means until he reaches the point where a default answer is triggered. This default answer is a statement that the bot was not able to find a proper answer, which means that before reaching this point is where the federation needs to act. In order to be able to give bots the capability of calling a federation, this runtime service was changed, and a new step was created that is called before the default answer.

The federation step will start by getting the bot properties so that it is possible to see if the current bot has the federation active and if there is any available slave. If the federation is configured, a new service called “Federation Service” will be called which is where the core features of the federation were developed. Having no federation configured is also an option and, in that case, the bot will continue to check every step until he reaches the default answer. This means that the federation has no interference on the original decision process if it is not configured.

### **7.5.3. Federation Service**

---

Chatbots with a federation configured will reach this service because it is where all the core federation features are configured. When a bot communicates with this service the first task that is going to be done is to see if the federation is for sure enabled, to avoid unnecessary calls that can slow down the bot’s decision process. Every unnecessary step or call will result in extra time getting the response and that is why it is always important to check if certain actions are necessary during the bot decision process. If the federation is enabled, the service will ask if there is already a bot selected to answer the question because when a user sends a message to a master bot and a delegation happens, the user will keep talking to the same slave until he wants to end the delegation. Given that it is the first time the bot fails to answer a user question, there is no slave available to answer so the service needs to find the best one to answer the question. To be able to find a slave that can answer there are two algorithms that search for a delegation among the configured mappings:

- NLU Search – Algorithm that queries all mapped slaves NLU system in order to find and pick the bot with the best confidence score;
- Text Search – Algorithm that compares the user input with the stored skills mapped for each bot.

The search algorithms implement an interface that will make it easier to expand the number of algorithms that can be used to decide which bot is the best to answer the question. Although NLU search can give better results because it asks directly each NLU system of each slave, it is important to notice that doing this process one by one can slow down the decision process. The fact that it is not the fastest way to search for a new bot it is used only after using the direct text comparison. Even though this is one of the reasons, the main reason the text algorithm is used at first is that if the configured system is, for example, Dialogflow, calling the service may have some costs associated and multiplying those costs for multiple bots will make it not suitable for federations with a large number of slaves configured.

After the decision process, the algorithm returns a mapping with information about what bot is going to be used from that point on as well as the channel that can be used to communicate with him. Using the channel's information, the message is sent to the slave. There are two channels available:

- Google Assistant – Used to send the message in order to get a message from google assistant;
- Federation Channel – Used to send the message to a bot that is available on the Wit Bot Engine.

There are two differences between these channels, one being the fact that is internal so gRPC was used for the communication while the other is an HTTP request, and the second being the fact that the federation channel is called asynchronously while Google Assistant is not asynchronous. The Google Assistant is not asynchronous because the NodeJs server, adapted from an open-source GitHub project [32], was created during research phase as a proof of concept and was not designed having asynchronism in mind. Since the Google Assistant is not a bot from the Wit Bot Engine, a model for the request and the response were created. Regarding the Google Assistant response, it is important to notice that the model is not the same as the Wit Bot Engine usual bot response, so a conversion needs to be done so that it is possible to return the response using the already available methods.

Both channels and search algorithms have modularity in mind, which means that if in the future if there is a need to create a new channel or a new search algorithm the effort to do so is minimal on the service's side.

## 7.5.4. Gateway Service

---

The gateway service is responsible, for the most part, to act as a bridge between the external conversation channels (for example, the Facebook messenger), and the chatbot that is connected to that channel. This service has an important role in this project because it is where the federation channel's logic is implemented and ready to be used, through a gRPC connection, by the other services. The gateway service is also responsible to manage the message flow from the moment the user message arrives until the moment a response is sent back to the user. The service is responsible to keep track of the conversation flow so if a certain user sends a message to a chatbot the response is sent back to the same user on the same conversation channel. In simpler words, the gateway service is the entry point for the user messages and the rest of the Wit Bot Engine infrastructure. This service is also responsible to receive the answer from the chatbot and transform the content based on the channel that sent the message. This step is important because even though the platform has a common model for the bot message types, the format of this message is not the same for every channel.

## 7.6. Federation channel

---

The federation channel is a GRPC endpoint, available on the gateway service that receives a message, in this case, from the federation service. This channel receives all the necessary information, from the message that needs to be redirected to the information about the bot that received the message and the bot that the message needs to be redirected to. This endpoint is specifically designed to be used by the platform services hence it is not expected, at least for now, to receive external requests explaining the decision of using GRPC, that is the default framework used to connect all the services, instead of a Representational State Transfer (REST) endpoint.

This channel is not only responsible for redirecting the message but is also responsible to do some validations using the configurations saved by the user on the federation channel settings page. The channel will not send the message to the slave runtime

if the leader bot is not on the whitelist or the slave is configured as a slave that can be used by every other bot. The message will also be rejected if for some reason the federation channel is disabled for that slave. Sometimes the bot's runtime service is down, which means that the bot is not available to process user messages and the communication between master and slave bot fails. If this happens the delegation process fails and the default message from the leader bot is triggered because it is the last possible step in a bot workflow. In case we are allowed to communicate with the slave bot and he is available we can then send the received message to the slave runtime. The message is sent to the slave bot messages queue to get processed, which means that the process is not blocked, and new messages can be sent. When a bot finishes the task, he sends the response back to the connector that is responsible to return the response to the federation service. The federation service is responsible to return the message to the master bot so the message can be delivered back to the user.

## 7.7. Context Variables

---

The context variables store information about the conversation essentially to keep particular pieces of information that may be necessary later on the conversation, such as entities. If, for example, the user wants to book a fight and gives the information about the location that he wants to visit, this location is one example of a piece of information that is stored on a context variable. The context variable can also store specific information about a bot, for example the bot's skills.

There are several context variables that can be used to take advantage of the several components available on the platform conversation builder to create a better master bot or a better slave bot. There are two new context variables available that were added into the platform to accommodate the changes that the federation project brings to the chatbots' flow. The context variables '*isUnderDelegation*' and '*stopWord*' are helpful to define specific flows for a slave bot.

With the *'isUnderDelegation'* variable, that stores the information about if the bot is being called by another bot, we can define a specific path for bots that are being used as slave bots by using the decision module. This can be used, for instance, to give a warning message to the user saying that the bot he is talking with is not the bot that he started the conversation with. This is important because even though we are using the bot as a slave bot, this same bot can be used as a standalone bot that in this case is not being called by another bot.

The *'stopWord'* variable is recommend to be used either for giving complementary information in a message, to keep the user aware how to end the delegation, or to be used as an option that can be clicked in order to automatically end the delegation when the conversation flow enters in a state in which the best decision is to return to the master bot. This may occur when the user changes the topic when he is already talking with a delegated bot, because the delegated bot is no longer able to answer the user's questions.

## 7.8. Optimizations

---

The federation service uses NLP services such as DialogFlow in order to check whether the bot is able to answer the user's input. Since it is used to search for a delegation it is also important to create a flow that reduces the number of times this selection process occurs. There are several ways to reduce the number of calls to the NLP service, but the simplest one is defining multiple start events. When there are multiple start events for the same flow it creates several patterns on the AIML file giving the user more than one possible sentence to trigger that flow. Having more than one start event for the same flow we can avoid the usage of external NLP systems because the user input can easily be matched using those start events because it uses the AIML engine trigger sentences and will have more room to use the text matching algorithm to find a delegation. The AIML engine used by the platform is called program AB [33] that is an implementation of the AIML 2.0 draft specifications. Preventing the usage of an NLP system by managing the conversation in a way that can rely more on using the AIML engine patterns is important because external NLP systems such as

DialogFlow can be charged when users exceed the quota. This is an example of how to optimize monetary costs, but we can also optimize the processing costs associated with the solution.

One way to optimize the processing cost when there is a federation mapped is by avoiding, as much as possible, to redirect the user to the wrong bot. If a user is sent to the wrong bot there is an extra bot processing the message, which means that it has twice the cost in terms of processing power used. This is extra processing that is unnecessary because at the end the delegated bot may not be able to answer anyway. This can be avoided by giving a higher confidence threshold on the mappings used on the federation. If the mapping has a higher confidence threshold, we only select another bot to assume the conversation flow if we have enough pieces of evidence that the bot will be able to answer the user input. Another way to optimize is by defining a general-purpose bot, such as Google Assistant, as default mapping. Having Google Assistant as default mapping will add an extra layer of reliability because Google Assistant is one of the most feature-complete bots and chances are that if none of the other bots can answer, the Google Assistant can. Another good practice is to use the bots with more knowledge so that the mappings list can be shorter. For example, if we have a bot that can answer questions about the weather and we have another one that can answer the same questions but can also answer about sports, it is preferable to use the second one. This will reduce the number of mappings the user has because if we want to later have access to knowledge about sports, we do not need a new mapping. If the list is not massive the algorithm does not have to iterate more than he needs to in order to find the correct mapping.

## 7.9. Facebook Personas

---

Facebook has a feature that is in a beta stage called Facebook Personas [34]. The main objective of this new feature is to allow, in the same chat windows, to change the avatar and the name from the perspective of the chatbot. In order to give users a better visual representation when using the Facebook Messenger channel to communicate with the chatbot, an initial implementation of this feature was done for the Wit Bot Engine platform. This gives the possibility to, when there is a delegation, show the image of the bot that the

user is now talking with as well as the name. This is a feature that is important from the point of view of a user that interacts with the chatbot because it enhances the user experience, in the sense that the user always has a better perception about which chatbot is answering is questions.

Facebook Personas is also a feature that aims to be used as a better representation of the state of the conversation in the sense that the user has visual cues about what is happening during the conversation. This feature has a clear advantage over the first versions of the project because when the chatbot has information that the user needs to know about the whole delegation process, it was a rather extensive text sent by the chatbot with all the necessary information. This extensive information is still being sent when there is a new delegation process but there is no need to send this text reminding the user which bot is answering is questions and how can he go back to the first chatbot.

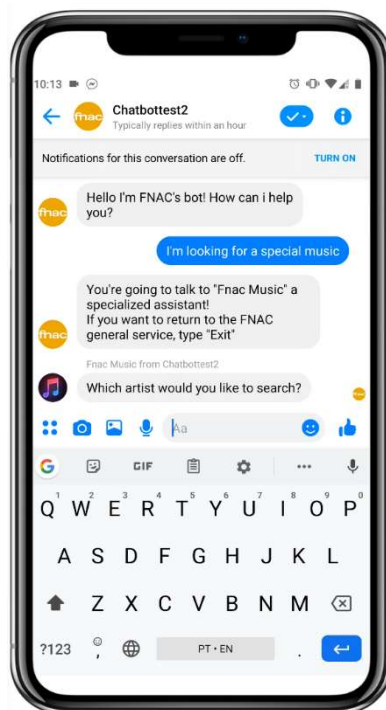


## 8. Demo

---

Every aspect of the project until this section is about the development of the platform itself and the necessary changes to the already existing Wit Bot Engine services so that the federation project could work as planned. Besides all that work, a simple demo was developed using the tools created during the development phase. The development deals with the creation of the tools that can be used by bot designers, which are responsible to create the behavior of bots using the Wit Bot Engine platform; the purpose of this chapter is to show the result from the perspective of a user that talks with a bot configured and deployed using the platform.

After all the configurations on the platform, a bot can be deployed in a conversation channel such as Facebook Messenger using a webhook. This webhook is used by Facebook to redirect the user input into the platform itself. This webhook always has an associated the bot name so that the platform can redirect the message to the proper runtime service where the message is processed in order to be able to get an answer out of it. Since there are new steps when the federation is configured, it will also influence the users that talk with the bots.



*Figure 12 - User texting chatbot with federation enabled*

This is particularly different when a user uses Facebook Messenger since Facebook personas were implemented. As can be seen in Figure 12, the user starts the conversation with a bot

that is a representation of a known store, and says “I am looking for special music”. Since the bot he was talking with does not know how to answer that question he sends a reply to the user to keep him aware that he is going to talk to another bot. When the delegation process occurs the second bot already knows what the user asked for and says “Which artist would you like to search” as it can be seen in Figure 12. Since the delegation process already

occurred the conversation will be handled by the same bot. If the user sends a message saying “I am looking for something to read on my vacations”, since the bot he is talking with is specialized in music and this bot does not have a federation, he send a message to the user informing that he should click “Exit” so that he can stop talking with the bot that knows about music. This message occurs when a bot knows that he is being delegated and, it knows this because there is a flag called “isUnderDelegation” that can be used to change the bot flow. This flag is set to true when the bot was called using the federation channel. As it can be seen in both Figure 12 and Figure 13, the user can clearly see that he is talking with

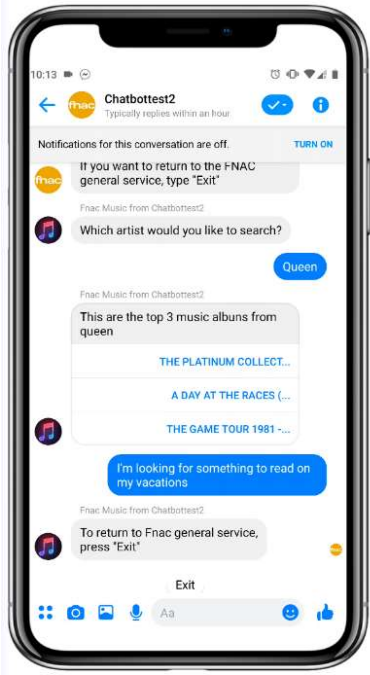
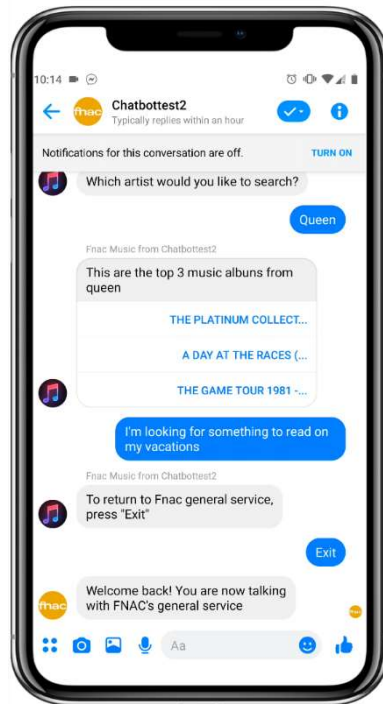


Figure 13 - Music bot default answer

another bot, not only because a reply is sent at the beginning of the delegation process but also because of the avatar and the name of the bot he is talking with changes. When the user either clicks on the button to call the stop word or says the stop word, the delegation will end and the user will return to the master bot. When the delegation process ends the user will



*Figure 14 – Delegation process finished*

receive a reply from the master bot so that the user is aware that he is not talking with the music bot anymore, as it can be seen in Figure 14.

This demo is an example of how a simple chatbot configured on the platform with a federation behaves.

## 9. Conclusion and Future Work

---

The fact this technology is in what can be considered to be an early stage, there aren't many alternatives on the market to compare it with so as to improve the solution. Even though the project was successful, particularly from a technical point of view, it is safe to say that some of the features should be worked on to improve several aspects of the solution – such as the performance and the chatbot users' experience. Since there are not many alternatives, it is also important to say that from the idealization of this project until the final result, some of the ideas about how the project should be implemented have changed. One the example is that the project started as an idealization of a true bot to the bot communication channel, and ended up being a project involving delegating work to other bots that eventually use a channel to share the user message between two chatbots in order to fulfill the task that the user asked for. Since this is not true bot to bot communication, it is not possible to chain them together to complement each other in order to, for instance, send a message to a Spanish bot that translates a message and then conveys the message to another bot that orders pizzas but that can only understand English.

There are other issues found along the way of the development process of this project – but not being able to predict how the users will behave is perhaps the biggest. Let's say that an user sends a message about music and the conversation flow is redirected to a bot that is specialist in that matter; if the user decides to talk about books, the user will “hit a wall”, and the best thing he can do is return to the bot he started the conversation with and ask about books again to be redirected to a chatbot specialized in music.

Since there are many ways in which the project can be improved, one of the top priorities from this point on is to improve user experience. User experience could be improved by adding, if possible, for every other conversation channel such as twitter and rich communication services (RCS), the behavior defined in the Facebook personas feature. Besides user experience, there are several other improvements that could be implemented, including:

- improving the algorithms that searches for a bot in the federation to avoid, as much as possible, using unnecessary NLP service calls;

- creating a more customizable interaction between the master bot and the slaves, in order to be able to create behaviors that cannot be achieved with the “one to one” communication approach
- the creation of more options of bot to bot channels, such as Facebook Messenger and Alexa if possible;
- creating a “is typing” behavior for multiple channels, to show that the bot is processing the user input and avoid frustration when there is slower response time from the chatbot as much as possible;
- improving the overall performance of the solution.

It is noticeable that most of the future work, in the short term, should be about improvements because seeing all the pieces working together gives a better holistic perspective. These problems, most of the time, are not visible in an early stage because some features can only be properly tested when all the necessary features are interconnected. For instance, if a function that is supposed to be called from another service is created, it will be harder to test it in isolation.

From my perspective, the main goals of the project have been reached successfully, not only because the necessary features were completely implemented, but also because it was possible to create a demo to showcase the project working properly. Even though improvements can be done, it is safe to state that the project was finished on time and it was integrated with the Wit Bot Engine platform successfully – which was the central point of this project.

# References

---

- [1] “WhatsApp is finally making a key move that will change how we use chat — Quartz.” [Online]. Available: [https://qz.com/596981/whatsapp-is-finally-making-a-key-move-that-will-change-how-we-use-chat/?fbclid=IwAR0OEYbso7hvVRw7D-6DSA2b20oVpj8DzgWoaaT\\_kM8Pb9YJDKULAXK0G2c](https://qz.com/596981/whatsapp-is-finally-making-a-key-move-that-will-change-how-we-use-chat/?fbclid=IwAR0OEYbso7hvVRw7D-6DSA2b20oVpj8DzgWoaaT_kM8Pb9YJDKULAXK0G2c). [Accessed: 10-Jan-2019].
- [2] “Facebook Messenger passes 300,000 bots | VentureBeat.” [Online]. Available: <https://venturebeat.com/2018/05/01/facebook-messenger-passes-300000-bots/>. [Accessed: 14-Nov-2018].
- [3] “80% of businesses want chatbots by 2020 - Business Insider.” [Online]. Available: <https://www.businessinsider.com/80-of-businesses-want-chatbots-by-2020-2016-12>. [Accessed: 23-Apr-2019].
- [4] “How to Build an NLP Engine that Won’t Screw up.” [Online]. Available: <https://labs.eleks.com/2018/02/how-to-build-nlp-engine-that-wont-screw-up.html>. [Accessed: 13-Nov-2018].
- [5] G. G. Chowdhury, “Natural language processing,” *Annu. Rev. Inf. Sci. Technol.*, vol. 37, no. 1, pp. 51–89, Jan. 2005.
- [6] “Chatbot Vocabulary: 10 Chatbot Terms You Need to Know.” [Online]. Available: <https://chatbotsmagazine.com/chatbot-vocabulary-10-chatbot-terms-you-need-to-know-3911b1ef31b4>. [Accessed: 05-Feb-2019].
- [7] M. das Graças Bruno Marietto *et al.*, “ARTIFICIAL INTELLIGENCE MARKUP LANGUAGE: A BRIEF TUTORIAL.”
- [8] Joost N. Kok, Egbert J. W. Boers, Walter A. Kusters, and Peter van der Putten, “ARTIFICIAL INTELLIGENCE: DEFINITION, TRENDS, TECHNIQUES, AND CASES.”
- [9] “Google Assistant, your own personal Google.” [Online]. Available: <https://assistant.google.com/>. [Accessed: 10-Jan-2019].
- [10] “Google Assistant SDK | Google Assistant SDK for devices | Google Developers.”

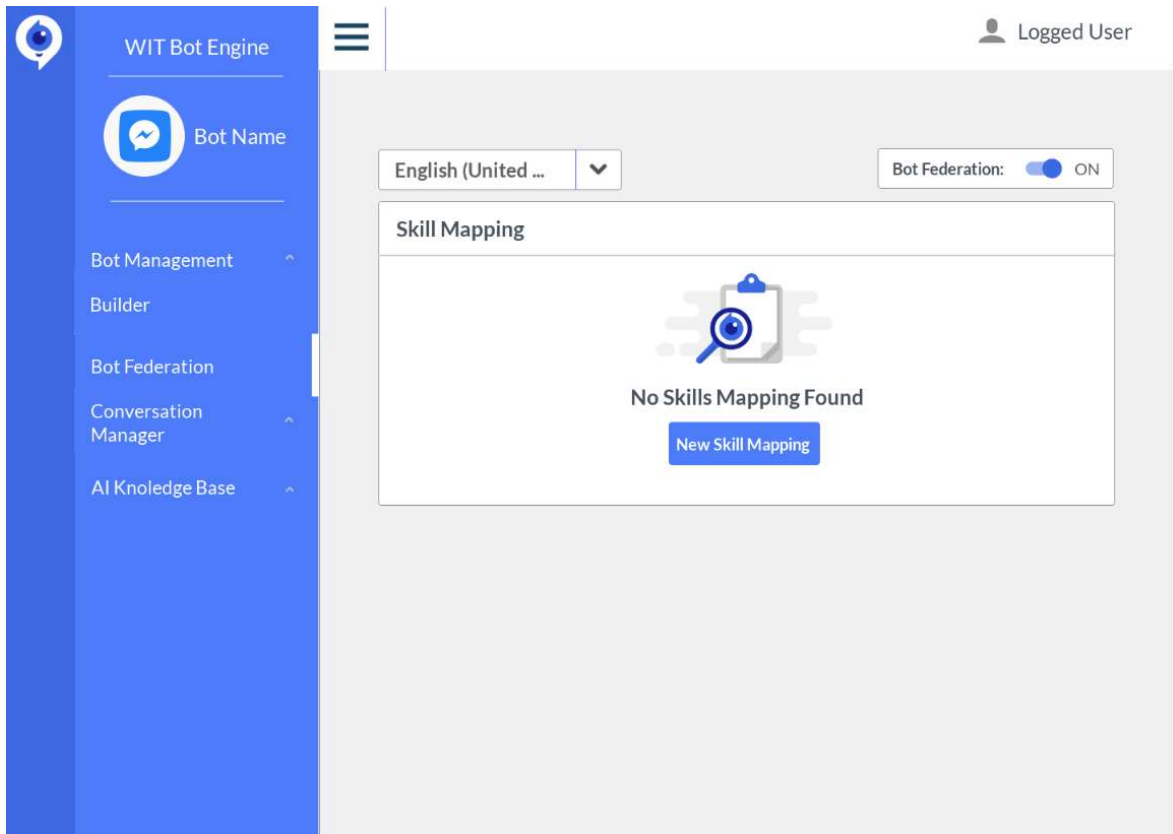
- [Online]. Available: <https://developers.google.com/assistant/sdk/>. [Accessed: 10-Jan-2019].
- [11] “Ways to Build with Amazon Alexa.” [Online]. Available: <https://developer.amazon.com/alexa>. [Accessed: 10-Jan-2019].
- [12] “Alexa Voice Service.” [Online]. Available: <https://developer.amazon.com/alexa-voice-service>. [Accessed: 10-Jan-2019].
- [13] “Siri - Apple.” [Online]. Available: <https://www.apple.com/siri/>. [Accessed: 10-Jan-2019].
- [14] “Snips — Using Voice to Make Technology Disappear.” [Online]. Available: <https://snips.ai/>. [Accessed: 10-Jan-2019].
- [15] “Mycroft – Open Source Voice Assistant - Mycroft.” [Online]. Available: <https://mycroft.ai/>. [Accessed: 10-Jan-2019].
- [16] “Messenger Platform.” [Online]. Available: <https://developers.facebook.com/docs/messenger-platform/>. [Accessed: 26-Oct-2018].
- [17] “Telegram Bot API.” [Online]. Available: <https://core.telegram.org/bots/api>. [Accessed: 25-Sep-2019].
- [18] “Business API.” [Online]. Available: <https://www.whatsapp.com/business/api>. [Accessed: 25-Sep-2019].
- [19] “Enable engaging conversations seamlessly across 30+ channels using a single API - Gupshup.io.” [Online]. Available: <https://www.gupshup.io/developer/home>. [Accessed: 10-Jan-2019].
- [20] “Dialogflow.” [Online]. Available: <https://dialogflow.com/docs>. [Accessed: 25-Sep-2019].
- [21] Dr. Richard S. Wallace, “The Elements of AIML Style.”
- [22] Bruce Wilcox, “ChatScript Documentation.” [Online]. Available: <https://github.com/ChatScript/ChatScript/blob/master/WIKI/README.md>. [Accessed: 26-Sep-2019].
- [23] “Artificial Intelligence Scripting Language - RiveScript.com.” [Online]. Available: <https://www.rivescript.com/>. [Accessed: 04-Jan-2019].

- [24] “codename botml | simple, modern and highly reusable syntax for writing powerful chatbots.” [Online]. Available: <https://codename.co/botml?lang=en>. [Accessed: 08-Jan-2019].
- [25] K. Schwaber and J. Sutherland, *The Scrum Guide*. .
- [26] “WIT Bot Engine • WIT Software.” [Online]. Available: <https://www.wit-software.com/products/wit-bots-platform/wit-bot-engine/>. [Accessed: 05-Feb-2019].
- [27] “Angular.” [Online]. Available: <https://angular.io/>. [Accessed: 08-Feb-2019].
- [28] “Spring Boot.” [Online]. Available: <http://spring.io/projects/spring-boot>. [Accessed: 08-Feb-2019].
- [29] “gRPC.” [Online]. Available: <https://grpc.io/>. [Accessed: 07-Sep-2019].
- [30] “facebook chat api.” [Online]. Available: <https://www.npmjs.com/package/facebook-chat-api>. [Accessed: 29-Sep-2019].
- [31] “Unit Testing Alexa Skills Using Virtual Alexa.” [Online]. Available: <https://bespoken.io/blog/unit-testing-alexa-skills-virtual-alexa/>. [Accessed: 26-Sep-2019].
- [32] “google-assistant: A node.js implementation of the Google Assistant SDK.” [Online]. Available: <https://github.com/endoplasmic/google-assistant#readme>. [Accessed: 26-Sep-2019].
- [33] “Program AB.” [Online]. Available: <https://code.google.com/archive/p/program-ab/>. [Accessed: 12-Jun-2019].
- [34] “Personas (Beta) - Messenger Platform.” [Online]. Available: <https://developers.facebook.com/docs/messenger-platform/send-messages/personas/>. [Accessed: 26-Sep-2019].

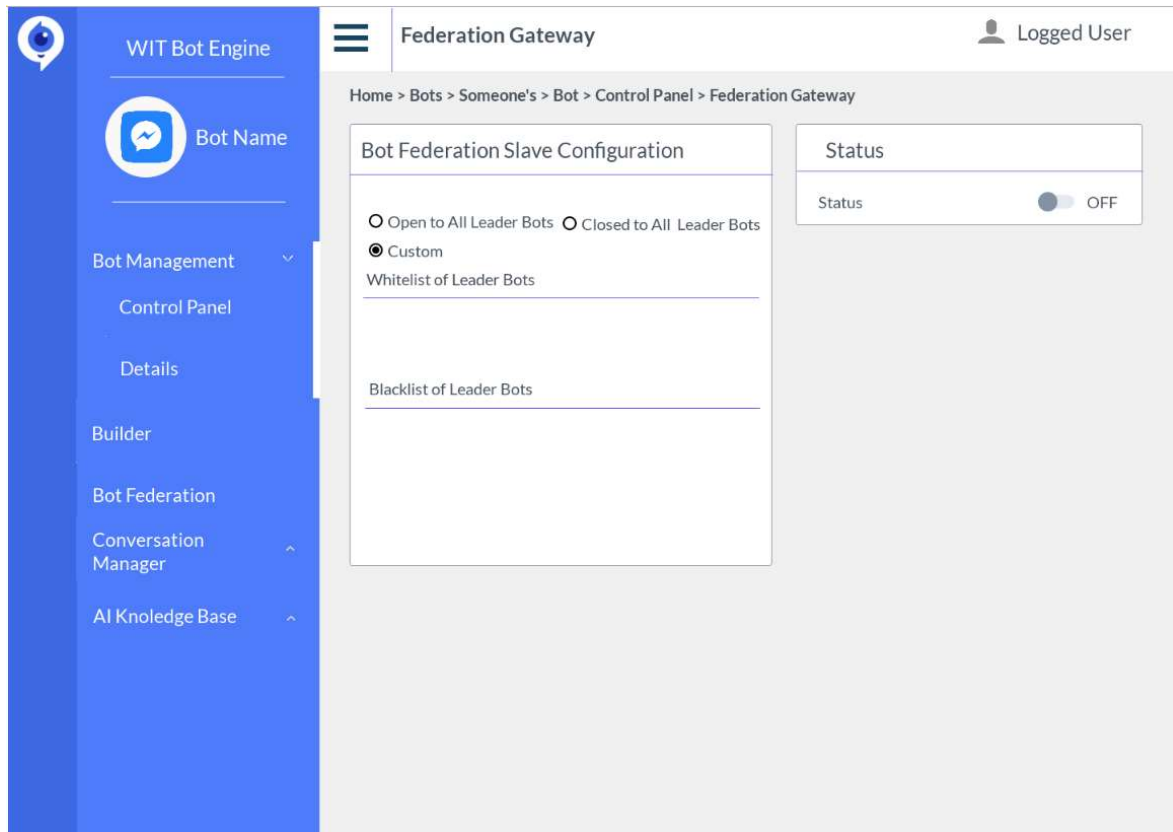


*This page was intentionally left blank*

# Appendices



The screenshot shows the WIT Bot Engine interface. On the left is a blue sidebar with the WIT Bot Engine logo and a menu containing: Bot Management, Builder, Bot Federation, Conversation Manager, and AI Knowledge Base. The main content area is titled "Skill Mapping" and includes a language dropdown set to "English (United ...)", a "Bot Federation" toggle switch set to "ON", and a central message: "No Skills Mapping Found" with a "New Skill Mapping" button.



The screenshot shows the WIT Bot Engine interface for the Federation Gateway. The sidebar is the same as in the previous screenshot. The main content area is titled "Federation Gateway" and includes a breadcrumb trail: "Home > Bots > Someone's > Bot > Control Panel > Federation Gateway". It features a "Bot Federation Slave Configuration" section with radio buttons for "Open to All Leader Bots", "Closed to All Leader Bots", and "Custom" (which is selected). Below these are text input fields for "Whitelist of Leader Bots" and "Blacklist of Leader Bots". To the right is a "Status" section with a toggle switch set to "OFF".

WIT Bot Engine Logged User

Bot Name

- Bot Management
- Builder
- Bot Federation
- Conversation Manager
- AI Knowledge Base

English (United ...)

Bot Federation:  ON

### Skill Mapping

Intent	Delegated Bot	Channel	Stop Word	Accuracy	Actions
News +2	CNN		Stop	-	
Meeting Rooms	Witty Bot		Bye	80%	
Default	Google Assistant		Goodbye Google	-	

+ Add New Skill Mapping

WIT Bot Engine Logged User


Bot Name

- Bot Management
- Builder
- Bot Federation
- Conversation Manager
- AI Knowledge Base

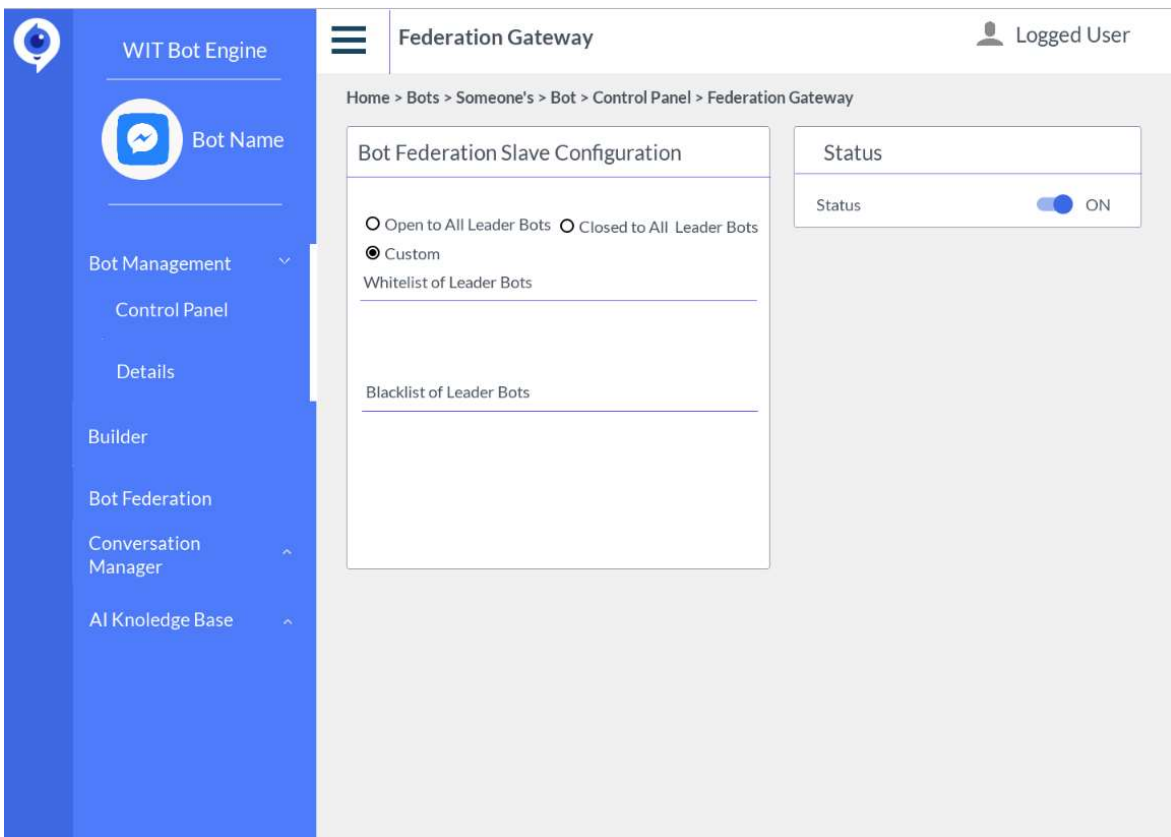
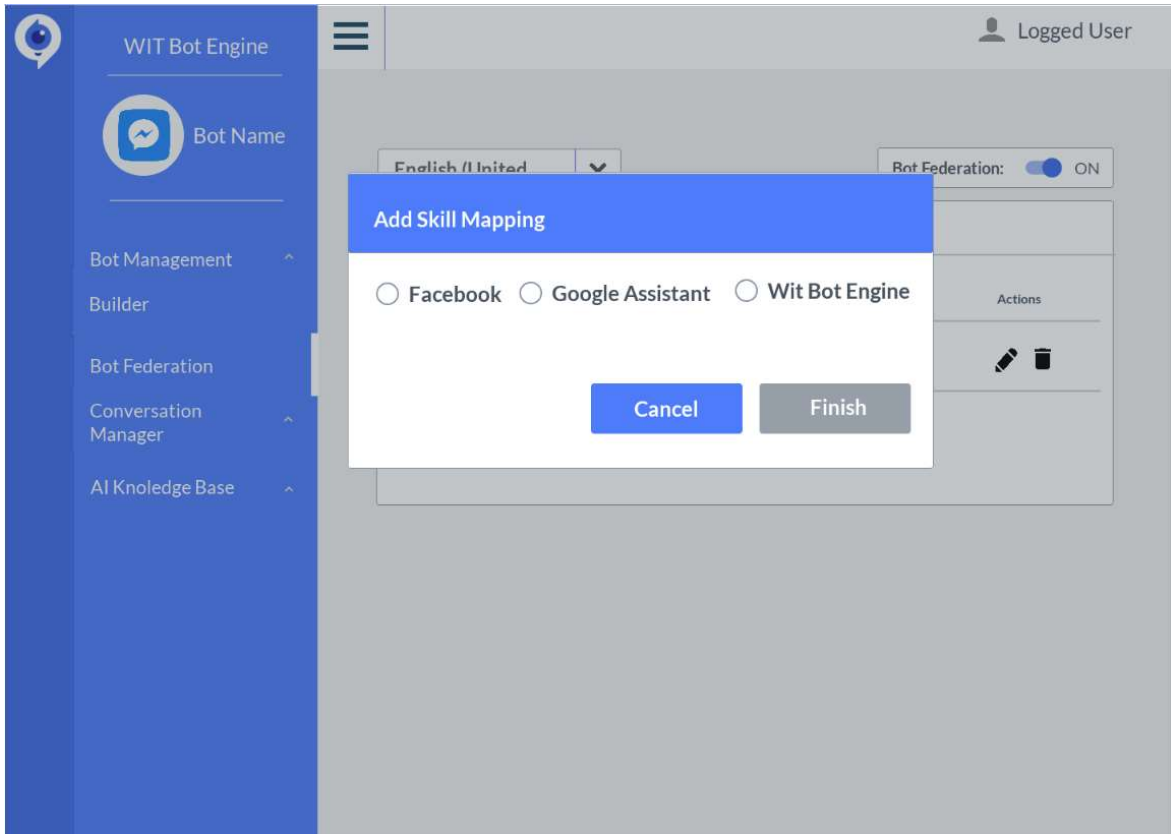
Select Language

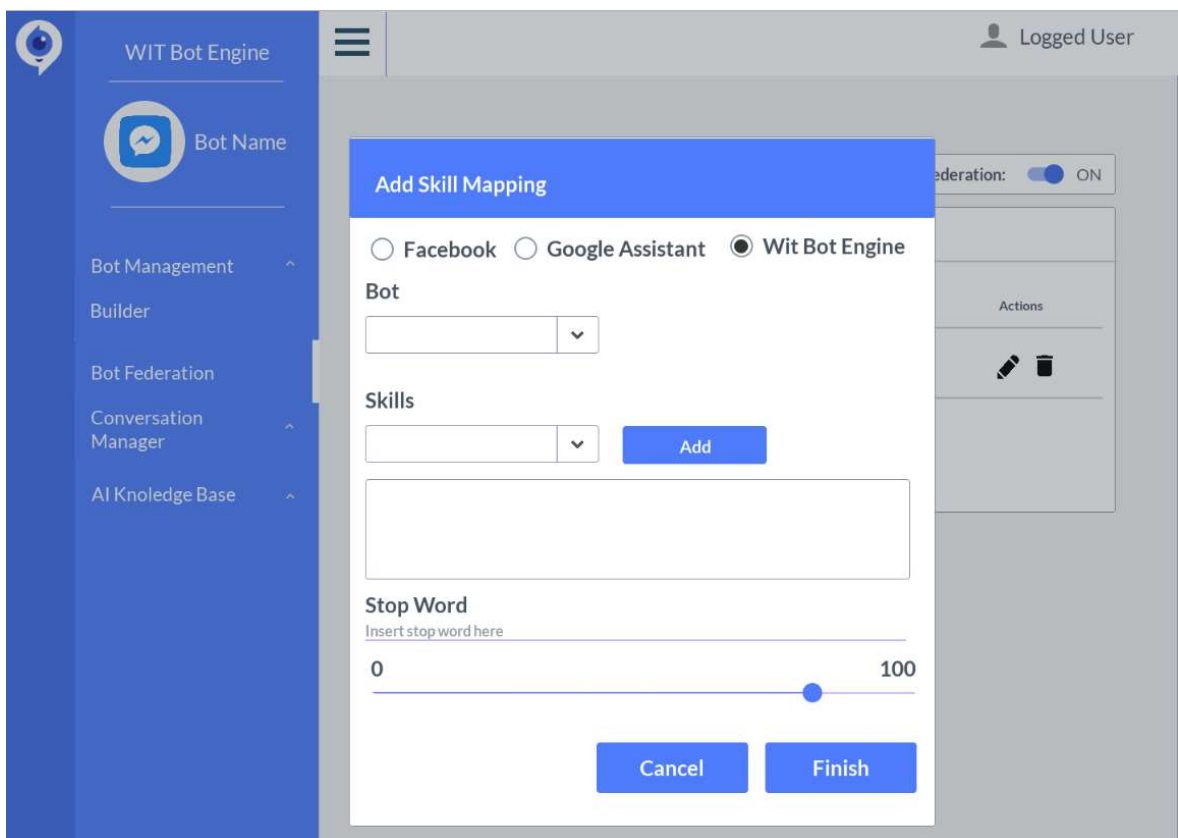
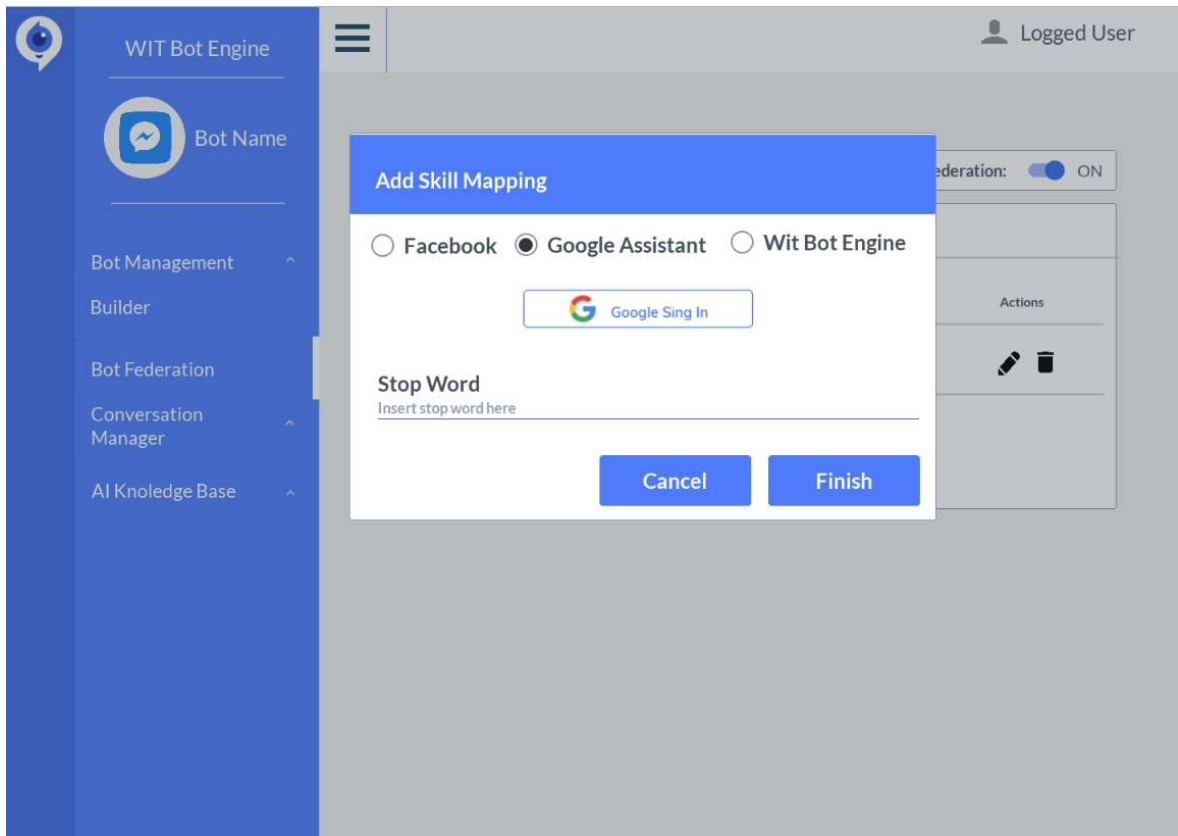
Bot Federation:  ON

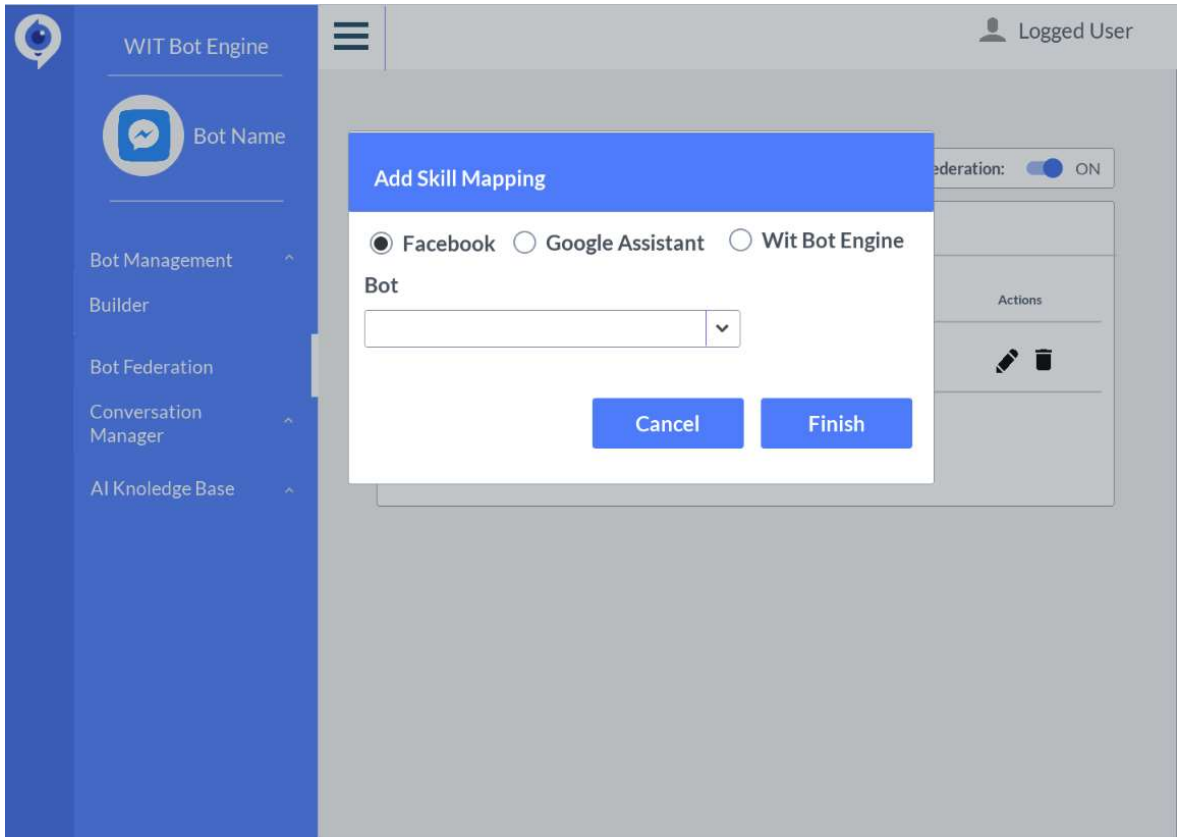
### Skill Mapping



**Please select a language**  
Select language in order to manage personas







*This page was intentionally left blank*