Raul Filipe Dias Barbosa

SENDING SERVICE INTENTIONS BY VOICE IN INDUSTRIAL ENVIRONMENTS

UNIVERSIDADE D COIMBRA

1 2 9 0

UNIVERSIDADE D
COIMBRA

Raul Filipe Dias Barbosa

SENDING SERVICE INTENTIONS BY VOICE IN INDUSTRIAL ENVIRONMENTS

VOLUME 1

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, supervised by Professor João Nuno Gonçalves Costa Cavaleiro Correia, Professor Tiago José dos Santos Martins da Cruz and Master Bruno Parreira, presented to the Faculty of Sciences and Technology/Department of Informatics Engineering.

September 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

Sending service intentions by voice in industrial environments

Raul Filipe Dias Barbosa

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, supervised by Professor João Nuno Gonçalves Costa Cavaleiro Correia, Professor Tiago José dos Santos Martins da Cruz and Master Bruno Parreira, presented to the Faculty of Sciences and Technology/Department of Informatics Engineering.

September 2021

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Abstract

Nowadays, we are used to using virtual assistants in Smart Home contexts or to ask for instructions. However, the potential that this technology presents has been growing, and as such, the first steps in applying these intelligent assistants in industrial environments are beginning to emerge. In this work, the main objective is to develop and implement the functionalities of a virtual assistant in an industrial context, which leads to the investigation of two distinct areas, policy management systems and virtual assistants. The integration of these two areas will enable the user to express its intention about the virtual assistant so that it, using policy management systems, can implement the network request. For all this logic to be successfully implemented, it was necessary to define, correctly and concisely, all the possible states that intent can reach in order to avoid inconsistencies in the system. To prove the operation of this system, in the end, a use case is presented about the creation of a service, where the user, using the virtual assistant, can create it. Once the specific knowledge in networks is not general knowledge, this technology has eased the user, allowing them to orchestrate the network through voice commands.

Keywords

Virtual Assistants, SDN, Policies, Intents, MyCroft, Networks

This page is intentionally left blank.

Resumo

Hoje em dia, estamos habituados a utilizar assistentes virtuais em contextos de Smart Home ou para pedir instruções. O potencial que esta tecnologia apresenta tem vindo a crescer e como tal, começam a surgir os primeiros passos na aplicação destes assistentes inteligentes em ambientes industriais. O principal objetivo deste trabalho visa o desenvolvimento e implementação de funcionalidades de um assistente virtual em contexto industrial, o que leva à investigação de duas áreas distintas, sistemas de gestão de politicas e assistentes virtuais. A integração destas duas áreas, vai possibilitar ao utilizador exprimir a sua intenção sobre o assistente virtual, de modo a que este, recorrendo a sistemas de gestão de politicas, consiga implementar o pedido na rede. Para que toda esta lógica seja implementada com sucesso, foi necessário definir, de forma correcta e concisa, todos os possíveis estados que um intento pode alcançar, de forma a evitar inconsistências no sistema. Para comprovar o funcionamento deste sistema, no final, é apresentado um caso de uso sobre a criação de um serviço, onde o utilizador recorrendo ao assistente virtual, consegue criá-lo. Uma vez que o conhecimento especifico em redes não é de conhecimento geral, esta tecnologia veio trazer facilitismos ao utilizador, permitindo orquestrar a rede através de comandos por voz.

Palavras-Chave

Assistentes virtuais, SDN, Politicas, Intentos, MyCroft, Redes

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

API Application Programming Interface. 4–6, 8, 13, 14, 20, 30–32, 36, 37, 43

BGP Border Gateway Protocol. 6

BNSF Base Network Service Functions. 6

CNS Current Network State. 14

DDPG Deep Deterministic Policy Gradient. 13

DPG Deterministic Policy Gradient. 13

DQN Deep Q-Network. 13

DROM DDPG Routing Optimization Mechanism. 13

ECA event-condition-action. 9

IoT Internet of Things. 1

IP Internet Protocol. 35, 41, 42

IT Information technology. 1, 46

JSON JavaScript Object Notation. 7, 14, 15, 28

LSTM Long short-term memory. 13

ML Machine Learning. 1, 2, 4, 10–18, 20–22, 25, 46

MLRC Machine Learning Routing Computation. 13

NIC Network Intent Composition. 6

ODL OpenDaylight. xiii, 5, 6, 15, 17, 28, 32, 38, 43

ONOS Open Network Operating System. xiii, 6, 7, 13, 15

OSDF Open Software Defined Framework. 8

OVS Open vSwitch. 33, 39

PBNM Policies Based Networking Management. 2, 7, 15

PCA Principle Component analysis. 12

PCEP Path Computation Element Protocol. 6

PNSF Platform Network Service Functions. 6

PobMC Policy-based Managers Coordination. 9

QoS Quality of Service. xiii, 8, 11, 14, 32, 38

RNN Recurrent neural network. 13

SAL Service Abstraction Layer. 6

SDN Software-Defined Networks. 2, 4–8, 11–13, 15–17, 20, 21, 28, 32

SLA Service Level Agreement. 8

SLO Service Level Objective. 8

SVM Support Vector Machine. 11, 12

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

The exponential growth of technology has facilitated the way people/companies view their daily lives, making, for example, processes and experiences more efficient. Virtual assistants insertion in the human environment has further strengthened this issue, ensuring that consumers' lives are even more straightforward and more practical [5]. In business terms, digital transformation is increasingly present, and some consider this a promising element for this transformation to occur. So investing in this sector is investing in the future of business [4].

Besides virtual assistants, this project involves a network management system that, through high-level policies, implement and monitor the network with the help of Machine Learning (ML) algorithms. With Internet of Things (IoT) devices growth, traffic volume on the global network increases, affecting its performance. It is believed that 94% of companies will be using IoT devices by the end of 2021 [28]. In the presence of this problem, something must be done to solve it. Policy management systems have the advantage of allowing the implementation of network functionalities to correct problems like the one described. In partnership with the virtual assistants, these systems function as a virtual Information technology (IT) department where, through natural language, users can implement rules on the network without requiring knowledge of it. These rules are applied through intentions, a type of policy that expresses objectives without mentioning how they are implemented. The virtual assistant will capture the user's intentions and transmit them to the network management system, implementing them.

1.1 Goals

This dissertation aims to implement features for a voice assistant in the context of industrial environments. The system will have to capture the user's intentions and implement them in the network. Simultaneously, the network is monitored to optimise to avoid problems such as policy conflicts or congestion. Thus we can expect results, such as:

- Development of ML models to identify the user's intentions through voice commands
- Development of user intention mapping mechanisms for analysis and optimisation of private networks
- Integration of the voice assistant with external components in order to demonstrate the developed functionalities

1.2 Contributions

The dissertation is part of an internal Capgemini project, where the system will be implemented. Part of this work (thesis) received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017226 – Project 6G BRAINS.

1.3 Document structure

The document's structure is divided into five sections, state of the art, methodology, system design, framework, and last experimentation.

The second chapter will present a technological review more oriented to the surrounding technologies. A literature review on the themes of Policies Based Networking Management (PBNM) and ML applied to Software-Defined Networks (SDN) is also undertaken.

In the third chapter, the methodology chosen for developing this project and its planning will be presented.

The fourth chapter starts with the requirements and architecture of the system. Then it presents some essential diagrams for the elaboration of the project and the use cases.

The fifth chapter presents the framework, the selected technologies, and all the steps implemented in this project. Finally, it ends with the implementation of the use case.

For the end, the sixth chapter presents the tests performed on the system.

This page is intentionally left blank.

Chapter 2

Technologies, Literature and Related Work

This chapter introduces the research done to achieve the objectives mentioned in the introductory chapter. In the first phase, the concept of Software-Defined Networks (SDN) is introduced, what this technology consists of and examples of two tools, concluding with related work on policy systems that use this technology. Next, we present concepts of Machine Learning (ML), in order to give an introduction of the concepts that will be used in related work on ML applied to network data. Finally, continuing in the area of intelligent systems, we conclude with the virtual assistants, where examples of open-source virtual assistants are presented and the features of each.

2.1 SDN & Policies Based Networking Management

This section begins by introducing the concept of SDN, providing an understanding of how this technology works, the differences to traditional ones and this architecture. Next, two SDN controllers that were studied are presented, understanding how they work. Finally, this section is concluded with a literature review on Policies Based Networking Management.

2.1.1 SDN

SDN is an architecture designed to make a network more flexible and easier to manage [31]. This approach allows programming the network behaviour through central control, which communicates with software applications from the available Application Programming Interface (API). One benefit of working with this software is that it allows the whole network's dynamic management, whatever the complexity [32].

We can assume that an SDN network allows [33] :

- Cost reduction
- Control over smarter traffic
- Smarter networks

Also, four key functionalities distinguish this software from the traditional ones [34]:

- Centralised control
- Separation of the data and control plan
- Interfaces between control and data layers
- Network programming

As we can see in Figure 2.1, this software is divided into three layers: Application layer, Control and Infrastructure layer.

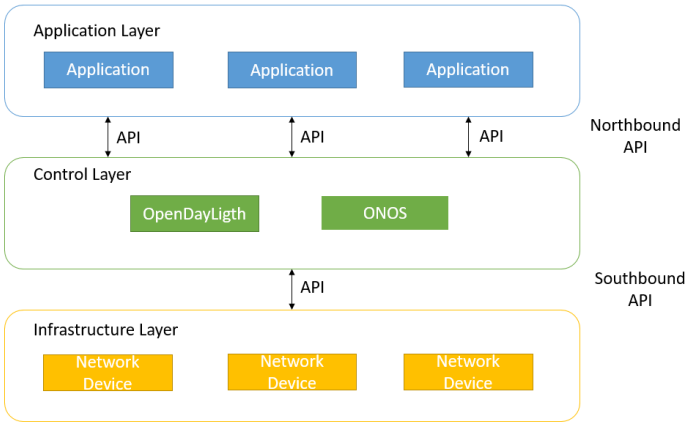

Figure 2.1: SDN System Architecture

All these layers are interconnected through APIs, in the case of the connection between the application level and the control, northbound APIs, and in the case of the control level and infrastructure, southbound APIs.

This software is advantageous for this project as it has a centralised controller and makes it easier to program the network, which are essential points for implementing the user's rules.

Before introducing the controllers studied, it is necessary to explain what intent is. An intent is an expression of the desired state that you want to be realised and can be considered [10]:

- Portable - can be moved between the different controller and network implementations and remain valid;
- Abstract - must not contain any details of a specific network;

The advantage of an Intent is flexibility, as it allows users to express policies using concepts and terminology that are familiar to the user without having specific knowledge in the field.

Introduced the definition of intents, we will focus on controllers, where two platforms, OpenDaylight and ONOS, were studied.

OpenDaylight (ODL) is an open-source platform that acts as an SDN controller to manage, program and automates networks. The first version of this controller was Hydrogen, launched in the first quarter of 2014. The most current version is called Aluminium. This platform is implemented in Java, which brings a great advantage as it can run on any system that has this technology installed [22].

As for the architecture, ODL is divided into three layers, as we can see in Figure 2.2

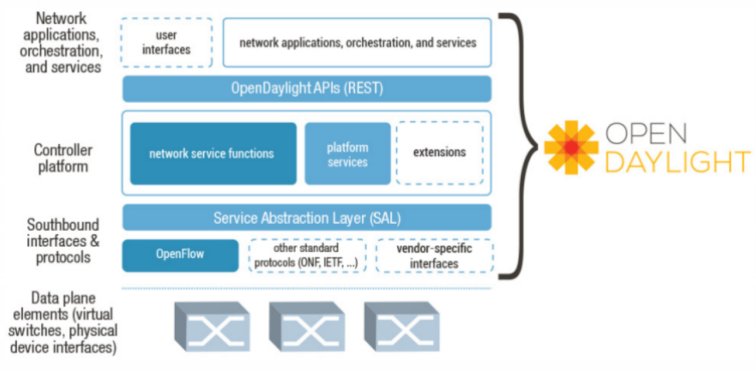

Figure 2.2: ODL Architecture, adopted from [22]

The control platform is the central part of this architecture, and this layer is responsible for understanding the Base Network Service Functions (BNSF), Platform Network Service Functions (PNSF) and Service Abstraction Layer (SAL). This layer also enables communication with external applications so that networks can be programmed through the northbound [22].

The communication interfaces created by the southbound protocols are in charge of maintaining secure and efficient communication. This support provided by ODL is done through plugins such as OpenFlow, Border Gateway Protocol (BGP), Path Computation Element Protocol (PCEP) and NETCONF [22].

In the application layer, we find the services and applications that monitor the controller and the network. We can find algorithms for the most diverse things in this component, from analysis to traffic to rule implementations. This communication is guaranteed through APIs, as mentioned before.

Since one of this study's objectives is to implement policies through user intentions, we must look at what this technology offers us. Network Intent Composition (NIC) provides the controller with the ability to obtain tools to manage and control network resources through intentions or policies. These intentions are obtained in the controller through the northbound interface, which, instead of traditional flow rules, offers more general and abstract semantics [21].

Open Network Operating System (ONOS) offers a control layer for a SDN network, providing control tools to manage network components and run software to provide communication services [23]. The ONOS platform includes [24]:

- Set of applications that act as extensible, modular and distributed SDN controller
- Configuration, management and implementation of new software, hardware and services
- Architecture in scale

The kernel, central services and ONOS applications are written in Java, as is the ODL.

As we can see in Figure 2.3, applications communicate with high-level intentions for what they program hardware. The northbound interface deals with applications and resolving

policy conflict and in turn, implementing them. The Distributed Core takes care of performance and management issues. As for Southbound, it is responsible for the structure and configuration of the network [30].

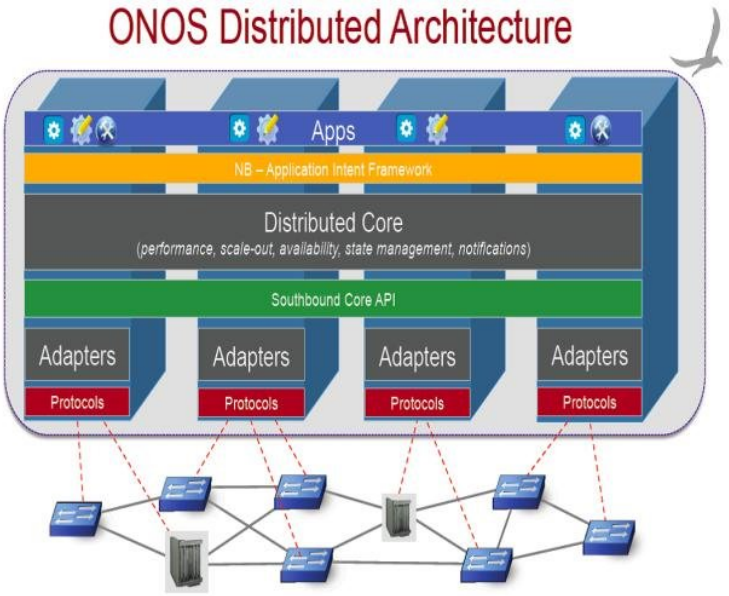

Figure 2.3: ONOS Architecture, adopted from [30]

When it comes to intents, ONOS has a subsystem called Intent Framework. This subsystem allows applications to indicate network control requests in the form of policy mechanisms. Although this tool already provides a set of predefined intents, the framework is designed to add other intents [25].

After analysing the controllers presented, the study related to Policies Based Networking Management follows how these systems work.

2.1.2 Policies Based Networking Management

Policies are like a set of rules and services that define the criteria for access and resource use. Each rule is composed of a set of conditions and their actions. The first set defines when the rule is applicable, and as soon as it is active, it generates a set of actions to be implemented. Policies Based Networking Management (PBNM) works as a manager that separates the rules that control the system from its functionality. As a result, this system reduces maintenance costs as well as improves runtime flexibility [26].

The research done by the author Vijay Varadharajan, present in [38] is constituted by an architecture based on policies in SDN context using the ONOS controller. This architecture is run on the SDN controller as an application. The policy servers connected to the respective controller consist of five main components: Repositories, Policy Manager, Policy Evaluation Engine, Policy Enforcer, and Handle Creator. The repositories consist of two sets: topology and policy repository. The first contains all network topology information, and the second stores all policies in expression form. These expressions are stored in JavaScript Object Notation (JSON) modulation language and specify a range of attributes associated with flow and the entities of SDN, such as Flow Attributes, Autonomous System Domain Attributes, Switch Attributes, Host Attributes, among others. The Policy Manager is the main component of the policy manager and, as the name suggests, manages each operation of the system. Policy Evaluation Engine is responsible for

assessing network traffic and checking which policies are related to flow. If policies need to be implemented, the policy enforcer is responsible. Finally, the Handle Creator creates the necessary handles for the controller using the policy manager. The purpose of these handles is to check the package's authenticity and apply the policies on the network.

In Machado's work [13], the author introduces a policy authorisation framework for SDN using a high-level language. This study focuses on policies targeted at the Quality of Service (QoS). By interpreting Service Level Agreements (SLA), the system extracts Service Level Objectives (SLO), which are considered requirements for assessing the network's performance. This framework is composed of two levels, Business and Infrastructure, where the first write the SLA in natural language and intentions and the second specifies the technical characteristics of the services. At the business level, we find components such as Policy Analyser, which uses natural language processing that, through regular expressions stored in the policy repository, matches the high-level expression, suggesting classes of QoS that most identify with their SLA. The refinement process of this system is divided into two phases, Bottom-up and Top-down. In the first phase, the process collects information from the network through OpenFlow, i.e., the best configurations for the SLA. With this collection, it is possible to frame which policies the high-level operator can apply. The second phase converts the high-level objectives extracted from the SLA and translates them into achievable objectives, SLO. If the operator does not accept any of the system's suggestions, creating policies and implementing them is also possible.

Douglas Comer and Adib Rastegarnia presented a framework, Open Software Defined Framework (OSDF) [7]. This system offers a high-level API that allows network users to configure and monitor the network to provide QoS. In addition, OSDF has mechanisms to analyse conflicts between policies to prevent two or more policies from conflicting when applied to the same targets. This framework's architecture consists of four components: Policy Storage Module, Policy Conflict Detection Module, Policy Parser Module and High-level network operation services. The first component configures and monitors the network based on high-level policies and consists of several services: Flow Rule, Topology, Region and Configuration. Policy Storage Module is responsible for storing network policies that can be listed, updated and removed. Policy Parser Module is responsible for reviewing policies and generating a set of flow rules. This module consists of three sub-modules: Path Selection Service, Traffic Selector Builder and Network Regions Parser. Finally, Policy Conflict Detection Module detects conflicts between active policies. This module is composed of sub-modules: Policy Conflicts Detection Service and Policy Conflicts Recommend Service. This study has a chapter dedicated to the detection and solution of conflicts since it is essential. As for detection, the authors use details from flow rules such as layer 2-4 addresses, action and priority. Through these details, the conflicts have been classified into five types:

1. **Redundancy**: a r_y rule is redundant if there is another r_x rule that corresponds to the same packet space, i.e. a more general r_y rule with the same action.

2. **Shadowing**: a r_y rule conflicts with r_x if r_x matches the same packet space as r_y but has a different action and a higher priority than r_y.

3. **Generalisation**: a r_y rule conflicts with r_x if r_x matches a subset of r_y and r_x has a higher priority and a different action compared to r_y.

4. **Correlation**: a r_y rule correlates with another r_x rule if r_y packet space intersects with r_x packet space, but r_y action differs.

5. **Overlap**:a r_y rule overrides another r_x rule if r_x matches the same address space as r_y and both its actions are the same.

The authors have implemented a high-level recommendation algorithm to advise network managers to resolve conflicts as far as conflict resolution is concerned. Each type of conflict has its solution:

- **Redundancy**: remove the policy with the lowest priority and whose address space is a subset of another policy.
- **Shadowing**: remove the shadowed policy, has the lowest priority, and the address space is a subset of another policy.
- **Generalisation**: remove the policy that has a more general address space and update the address space conditions.
- **Correlation**: update the policy address space by lower priority, remove the standard items, and insert the updated policy into the system.
- **Overlap**: remove both policies from the system and then insert a new one composed by merging both address spaces.

A system of policies called Policy-based Managers Coordination (PobMC), was created in the [15] study. The system policies are formulated according to the event-condition-action (ECA) rule. As long as an event occurs, there is an action taken as long as the condition is true. This framework is essentially made up of five elements:

1. **Policy Refinement Process:** this process is responsible for elaborating a graphical structure of the high-level objectives that the system can address. This process checks each type of policy during policy execution where the priority of execution is analysed to reduce potential errors.
2. **Dynamic Conflict Resolver Process:** this process's role is to decide to authorise each policy and apply logic to reach a decision. It checks the state of the system to see if it is necessary to apply the action. "if condition then Action"
3. **Policy Verification:** this process is dependent on the logical separation of adaptation and functional logic. With this, the adaptation layer is verified without the actor layer being verified.
4. **Context Monitor:** this process's objective is to monitor the environment, verifying structural and behavioural changes.
5. **Self-Coordinator:** central component of the system that coordinates all actions during the execution time. All decisions made in previous processes are verified in this component.

In addition to these elements, a repository and policy compiler are found in this architecture.

The research presented in the previous study [15] was further developed on the issue of conflict management between policies. Abdehamid Abdelhadi Mansor presented a statistical analysis technique to resolve policy conflict [14]. Two classes of conflict were considered: static and dynamic. The policy compiler uses the first class to detect specification errors

and reduce runtime conflicts between rules where the event and condition are matched. The second class detects and controls potential conflicts between policies that are not detected in the previous class through runtime meta-information. Therefore, the conflicts were divided into four categories where for each category, we can find static or dynamic classes.

1. **Internal policy conflict:** if there is an incompatibility between policies associated with the same functions
2. **Foreign policy conflict:** when the combination of functions in isolation does not present conflict but contains policies that, in coexistence, conflict.
3. **Conflict of policy space:** when there is more than one policy space managing the same set of issues, exposing different policies.
4. **Conflict of functions:** when a user obtains a set of incompatible functions.

However, the authors focus on static policy analysis.

Based on these categories, the classification of several expected conflicts has been introduced into the system. **Conflict mode:** when there is a triple overlap between the set of actions and targets. All the subjects and targets with different policies to implement must be identified to resolve this conflict. **Inconsistency**: when there is a policy of obligation but no policy of authorisation. The subject, target and authorisation policy must be determined to apply each conflicting powers policy to avoid this conflict. **Multiple Manager's Conflicts:** overlapping areas related to resource sharing. To resolve this type of conflict, authorisation must be given to allow managers to implement policies if they cause no conflict.

After presenting the study on controllers and policy management systems, the concepts of intelligent algorithms in the context of networks data are introduced.

2.2 Machine Learning in network management

ML is an area of computer science that consists of building algorithms that can learn from data and generalize to unseen data. These algorithms work by building models from specific inputs in order to make choices without following static instructions [16].

This section focuses on a brief introduction to the area of ML and where it can be found in the context of networks. First, therefore, we will introduce some necessary concepts, which will be applied in section 2.2.2.

2.2.1 Machine Learning-concepts

Before classification models can be applied, it is sometimes necessary to pre-process the data, as they may contain, for example, redundant features not relevant to the study. When we analyse a dataset, we find a wide range of features, with differences in value ranges for each one. These differences can affect the model's prediction, so changing their values on a common scale is necessary. In section 2.2.2, standardisation algorithms are used to improve the model's performance. The standardisation process is sometimes not enough to get the best performance from a classifier. Some features are more useful for

classification than others, which leads us to another type of data pre-processing, feature selection. Since there are more important and decisive features for classification than others, we should select the smallest set of attributes needed to distinguish between class members and others outside the class. Selecting the type and number of features has a big impact on the effectiveness and performance of the ML algorithms [11]. This type of pre-processing is used in most of the ML articles mentioned in the section 2.2.2. Related to feature selection are feature reduction methods, which measure the correlation between features. Highly correlated features are redundant and add little to the performance of the classifiers. Eliminating such redundant features is important since it reduces computing costs without having a significant loss of information [2].

Once the data processing is complete, we can pass the classification where, for this context, the intelligent algorithms are divided into three categories: Supervised Machine Learning, Unsupervised Machine Learning and Reinforcement Machine Learning. These are the three areas mentioned in the articles studied 2.2.2 and are explained in the following paragraph.

Starting with Supervised Machine Learning requires a set of data with the assigned classes,i.e. the data need to be labelled. These models learn from the labelled data set, and then through these, they predict future events. There is a division between training and testing. For the former, data entry is a known data set with the corresponding labels, where the model learns through relationships between them to predict the test set then [17]. Unlike the first classification category, Unsupervised Machine Learning does not require a labelled data set or training set. The main objective is to explore the patterns between the data and predict the results. In this model, we supply the data to the system and ask for relationships between them, based on the characteristics, to group them to make sense [18]. Like Unsupervised Machine Learning, the last classification category does not need labelled data and works based on an agent's "education". The algorithm will penalise the agent if he fails and will reward him for performing the function. Reward and penalty are the basis of these models because they allow determining the agent's exemplary behaviour in a specific context [17].

2.2.2 Related Work

In this section, all research carried out in the context of ML applied to data from networks is mentioned. In addition, research has been done in the context of traffic classification and network optimisation.

Rating traffic is a crucial step when it comes to providing network QoS. By being able to classify, we become aware of the network traffic profile, and in turn, we can provide efficient and fast solutions [3].

In Raikar's work [27], the author introduces a system model that describes the integration of SDN with supervised ML models for classifying network traffic. The architecture of this system is composed of four layers: Decision and Control Layer, Feature Extraction Layer, Data Acquisition Layer and a common layer to all where the model of ML for traffic classification is located. The controller chosen for the system was POX, and the algorithms for the classification were Naive Bayes, Support Vector Machine (SVM) and Nearest centroid. These algorithms were applied to the training and testing set to classify network traffic and finally compared to the results. The network topology was created by mininet. The three classes studied to classify traffic was HTTP, mail and streaming. Tcpdump was the tool chosen by the authors to capture data from the network. In the pre-processing, the netmate was used to obtain the flow statistics and label. Some of the

study's features were source IP, source port, destination IP, destination port, protocol, total_fpackets, total_volume and total_bpackets. The classification results were positive, exceeding 90 per cent in the three models applied, highlighting the SVM and Naive Bayes.

As in the previous study, the authors, Zhong Fan and Ran Liu, present, besides supervised learning approaches, unsupervised algorithms to classify traffic [11]. The presented model classifies the traffic through its statistics, such as flow duration, packet length, segment size, times and port numbers. With this intelligent method of classification, the aim is to reduce computational costs. Applying the model was necessary to have a dataset composed of Web, Mail, Bulk, Services, P2P, Multimedia, Database and Attack classes. Before classifying these classes, the authors apply feature selection filters like the Information Gain Attribute Evaluation from WEKA. After selecting essential features, algorithms like SVM and k-means were chosen to classify. For each rating model, the performance was calculated using the F-measure formula. The model SVM obtained better results than k-means for the eight classes under study. Even when new data sets were inserted, the accuracy for Web, Mail and Services remained above 90 per cent, however Bulk, P2P and Database had a steeper decline from the previously rated test set.

An application SDN for network traffic monitoring and classification, using algorithms of ML, is presented by Pedro Amaral [2]. This application collects data through an OpenFlow switch, which uses mirrors to capture data and send it to the controller. The data set used was tagged and generated under conditions controlled by a host. The features chosen for classification were: Packet size, Packet timestamp, Inter-arrival time, Source and Destination Mac, Source and Destination IP, Source and Destination Port, Flow Duration, Byte Count and Packet Count. In addition, it was necessary to normalise the data because it has very different characteristics with values in different scales. For this purpose, StandardScaler was the chosen algorithm. Another critical point was the correlation between features, and for that, it was chosen Principle Component analysis (PCA), to find the main components. The classification algorithms present in the study are supervised, namely: Random Forests, Stochastic Gradient Boosting and Extreme Gradient Boosting. The results were quite encouraging and similar among the three types of models, with results above 90 per cent for some classes. With this, it can be concluded that data obtained through SDN mechanisms are adequate for traffic classification.

A method of ML to classify the network traffic in partnership with SDN architectures is presented by Mohammad Reza Parsaei [29]. Since the network switches are connected to a central controller, the traffic classification protocol is identified based on the application layer. The controller used by the authors was the Floodlight. This system is compassed of two phases:

1. Offline phase: This phase is responsible for collecting data on the controller and for building the training set and model
2. Online Phase: Through the data from the offline phase, it classifies the network traffic.

As for the selected features in the offline phase: Source and Destination IP, Transport Layer port in Source and Destination, Transport Layer protocol in flow and backflow. The chosen algorithms for the online phase traffic classification are Feed-forward Neural Network, Multilayer Perceptron, Naïve Bayes and Levenberg-Marquardt Algorithm. The study classes for this system were FTP, HTTP, Messages and Streaming. As for the previous classes' accuracy, the algorithms showed promising results, above 95 per cent, with the Naïve Bayes algorithm standing out with 97.6 per cent.

Once the study for traffic classification is concluded, the research on Network Optimization in SDN begins.

The system, Machine Learning Routing Computation (MLRC), based on SDN is created by Sebastian Troia, in order to apply models of ML to optimize network configurations [37]. The controller chosen for this system was the ONOS. REST APIs capture traffic matrices every 5 seconds and train the model to keep it updated. This model aims to classify these matrices using a supervised algorithm which in turn is trained with optimal routing solutions. These solutions were obtained through the Net2Plan tool. The architecture of MLRC is divided into four sub-modules:

1. Data collection: Every 5 seconds are extracted from the traffic matrix switches.

2. Model training: The system's main component, where the algorithm is trained with the help of the Net2Plan tool. In this sub-module, the model learns how to classify traffic matrices that share the same routing configuration. The algorithm chosen was Logistic Regression.

3. Classification model: Responsible for classifying the traffic matrix. The output of this module is the ideal routing.

4. Routing computation: Converts the classification model output into flow rules to implement on switches.

This whole process is done in about 80 milliseconds, which leads us to conclude that we can apply ML to SDN to optimise it.

In Changhe Yu work [39], the author introduces a framework that uses a new mechanism based on deep reinforcement learning, Deep Deterministic Policy Gradient (DDPG) to optimise routing in SDN. This framework uses a mechanism, DDPG Routing Optimization Mechanism (DROM), to perform global control and management. Agents interact with the environment through three signals: state, action, and reward. Thus, by receiving the state of the network from the SDN, it activates the agent responsible for decision making and takes on a strategy, and in turn, the corresponding policies are implemented. The main objective of the DROM is to find the optimal action 'a' according to an entry state 's' to maximise a reward 'r'. With this, it is possible to optimise the maintenance and management of the network. The type of ML chosen by the authors was policy-based reinforcement learning,DDPG, where there was a combination of two methods, Deep Q-Network (DQN) and Deterministic Policy Gradient (DPG). DDPG generate strategic functions through neural networks and form an efficient and stable control model. This system's results have shown good convergence and effectiveness compared to existing routing solutions, which leads us to conclude that it is a good solution to be applied.

Neural networks are used by the framework, Neurote [6]. This framework comprises three essential modules: Traffic Matrix Estimator, Traffic Matrix Predictor and a Traffic Routing Unit. The first component is responsible for defining the traffic matrix and the interfaces with the rest of the components. The second component is in charge of estimating future network traffic from past and present traffic data. This component uses neural networks Long short-term memory (LSTM) and Recurrent neural network (RNN) where for each algorithm, the data is divided into training phase and test phase. The training phase is supervised and uses the backpropagation algorithm, where weights are changed until the error falls below the chosen threshold. The test phase focuses on the choice of optimal routes based on the previously predicted matrix. This component is based on supervised

learning using Deep Feed Forward Neural Network to combine traffic requirements with routing paths.

Supervised learning models are the central resource of the framework created by the author Li Yanjun [12] to propose an optimal routing solution. This system's architecture comprises three layers: Input, Dynamic routing, which contain the system logic and output. The input consists of the network topology information and the Current Network State (CNS) and QoS. The output is the ideal path from the source node to the destination. As for the main layer, a heuristic algorithm is built to route the flow through the minimum load path based on the network's current state. This algorithm is used to train the neural network and thus provides heuristically similar results directly and independently. The implemented ML approach achieved almost as good results as the heuristic algorithm. However, it showed faster times when executed.

Once the study about the area of ML applied to network data and policy management systems is finished, it remains to be seen how to capture the user's request. For this purpose, the investigated virtual assistants are presented.

2.3 Virtual Assistants

There are several types of virtual assistants around us. Today the name Alexa or Siri leads us to think almost automatically about voice virtual assistants. This is the type of virtual assistant chosen, for which more in-depth research will be done and consequently implemented in the system. Since it will be necessary to develop new features on the system, we have to look at open source assistants.

SUSI AI [36] is an open-source virtual assistant capable of interacting with the user through voice, using a API. This virtual assistant allows us to add more features to allow the user greater control over it; that is, it allows us to add, edit and remove skills. This type of assistant supports Linux, Android and iOS and can be integrated with speakers and vehicles. SUSI has its language; however, JavaScript can be a possibility to manipulate the assistant. As for dedicated devices, SUSI does not lack any yet. It also allows to transform the user's data into JSON format and manipulate it according to its intention.

SEPIA Framework [35] means server-based, extensible, personal and intelligent assistant that consists of a Java server and a client that can run on various platforms such as Android, iOS, Windows, Linux and Mac. The server is based on the REST architecture, and the clients use the HTTP protocol to communicate. It is on the server that the understanding of natural language, dialogue management and intention is done. The client handles speech recognition and converts the voice into text, sending it to a SEPIA server to interpret and present the result to the user through text, which can be in JSON. SEPIA and the already implemented services allow us to create our commands like the previously spoken technologies.

Of the virtual assistants studied, the most relevant was Mycroft AI [20]. Mycroft AI is an open-source virtual assistant that allows modifying, create and view code. This assistant gives the user freedom to have control over the system. Unlike other technologies like Alexa, Siri and Google Home, it only captures the voice after triggered. This virtual assistant can be found on various systems, from Raspberry Pi, Windows, Android and Mac. Besides these platforms, there are also dedicated devices, Mark 1 and Mark 2. Mycroft works by intents, once awakened, the user expresses the intention before the system, trying to interpret the intention and find the appropriate Skill. These abilities can be installed or

removed by users and can be easily updated to expand functionality. In addition to these advantages, Mycroft has the particularity of transforming the user's request into JSON format, which is useful for implementing the system. The Adapt Intent Parser is an open-source software [1] that creates a data structure through natural language. This structure is composed of three parameters:

1. User request
2. Probability of trust
3. Entities involved

An example of the structure created can be seen in Figure 2.4.


```
{
    "confidence": 0.61,
    "target": null,
    "Artist": "joan jett",
    "intent_type": "MusicIntent",
    "MusicVerb": "put on",
    "MusicKeyword": "pandora"
}
```


Figure 2.4: Adapt intent parser - data structure, adopted from [1]

This structure facilitates the way that we identify user intent through voice commands.

2.4 Conclusion

This chapter presented an investigation about the possible technologies/tools to be used to develop the project and explained some important concepts. Starting with the SDN controllers, after introducing the concepts, we realised that this technology brings advantages concerning traditional networks, once it allows to have centralised control, separate the control plane from the data and make the network programmable. Two solutions were presented, ODL controller and ONOS. Of these, the ODL controller stood out more for its feature offering and documentation. From the literature analysed on PBNM, I concluded that importance should be given to the pre-processing part of the policies, that is, validation and conflict handling, for the system to work properly.

In what concerns the area of ML in a network management context, some concepts were introduced, namely, concepts that were present in the research on traffic optimisation and classification. In this context, it should be highlighted that most of the models applied were supervised and also, when compared to the other types of models, they were the ones that obtained better results.

Of the open-source virtual assistants analysed, all of them presented similarities in characteristics. However, the one that offered better documentation and answered most of the problems was Mycroft [20]. Therefore, this virtual assistant was analysed in more detail and tested to validate it for use.

Chapter 3

Methodology & Work Plan

This chapter will present the methodology chosen for this project's development and all the planning done in the first and second semesters.

3.1 Methodology

For the elaboration of this work, the methodology follows the waterfall model. This model argues that activity should only start after the previous one has been completed and verified. However, it may be necessary to get some details right in earlier phases, which allows us to go back and modify what is necessary. The model begins with a study on the various themes addressed by this thesis as virtual assistants, controllers SDN, network managers using policies and ML applications to a data network, preferably using SDNs. After the search, all knowledge gained at this stage is applied to the implementation of the components. Finally, the methodology ends with the tests on the implemented system and the analysis of the individual test results, as we can see in Figure 3.1.

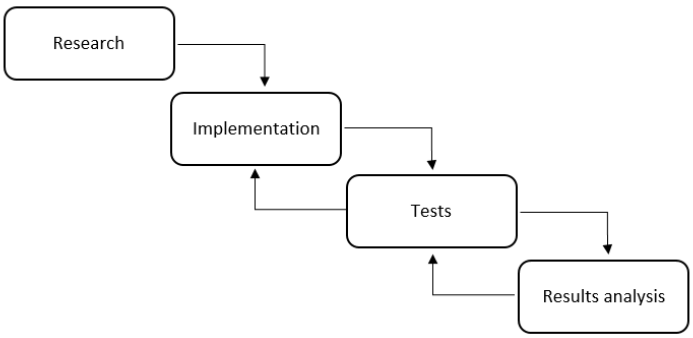

Figure 3.1: Methodology

3.2 Work plan

In this chapter, we detail the work plan created, which was divided into two semesters.

3.2.1 First Semester

During the first semester, the objectives focused on studying systems and technologies related to the subject of the dissertation. In an initial phase, research on policy systems and applications of ML to data from networks was done. Virtual assistants and SDN controllers were studied in more detail as they will be essential components of the system. Mycroft and ODL were the technologies selected for testing. For the former, the tests focused on creating skills and handling the user's request so that the output was in the desired modelling language. For ODL, together with the mininet, features were tested to assess whether the controller functions would be sufficient. As a preliminary work, traffic classification was the first topic to be advanced. Techniques of ML were applied to a data set in order to classify traffic. The report was followed over time, gaining more attention in the final straight. The Figure 3.2 demonstrates this whole process.

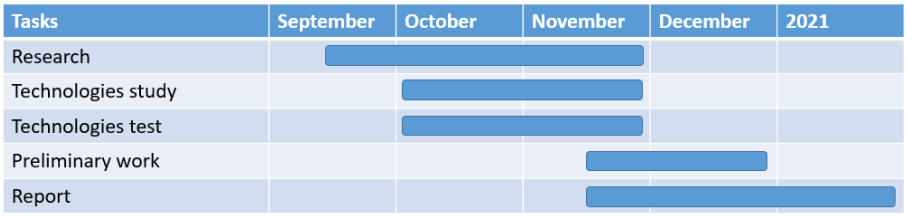

Figure 3.2: Work plan for the first semester

3.2.2 Second Semester

The planned work, Figure 3.3, and the executed work, Figure 3.4, are presented for the second semester. I adopted the strategy of presenting two graphs (planned and executed) because there were tasks that became more critical for the development of the project, which led to greater dedication of time.

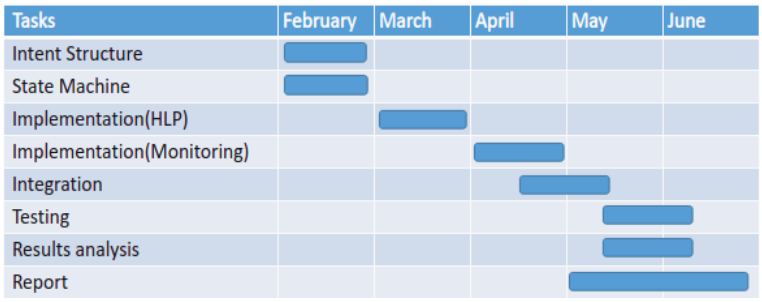

Figure 3.3: Work plan for the second semester-planned

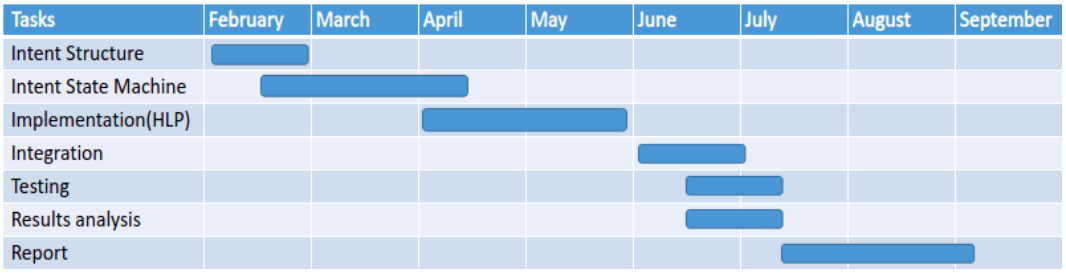

Figure 3.4: Work plan for the second semester-implemented

Initially, I started by defining the system design—this step started by defining the intents structure and life cycle (Intent State Machine). Compared with the time that I had planned for these steps, I ended up dedicating more time than expected, since both the structure of intents and the state machine would influence the operation of the system, and as such, it was necessary to investigate all possible cases to create concise and comprehensive structures. After defining the system logic, the implementation was changed, and we started to implement only the policy insertion phase in the system (Implementation(HLP)), leaving the monitoring phase for future work (Implementation(Monitoring)). This decision was made in agreement with the Faculty's and the company's Advisors since defining the previous steps correctly was essential for the rest of the project. In the implementation phase of the policy management system, we applied the logic defined in the system design (validation, conflicts, compilation and installation), leaving out the monitoring. Finally, integrating the system as a whole, we started the testing phase and the analysis of results. The report gained focus in the final phase of the project.

3.3 Conclusion

In the initial part of this project, everything went as planned with no big surprises. However, as we entered the second phase of the project, new ideas emerged that made perfect sense to create, and much time was spent on making the policy management system consistent. The intent modelling framework is an essential step for interpreting the system as it allowed us to define a concise structure that could adapt to any intent and interpret the request so that the system could understand it in a generalised way. In addition to this structure, the need arose to create a state machine representing the life cycle of intent from creation to installation and subsequent monitoring. Naturally, these structures suffered changes as I progressed, and with the help of the supervisors, the company and the university, all kinds of possibilities were discussed to make this state machine as complete as possible.

Dedicating time to system design ended up influencing the time that I had reserved for the monitoring phase where I would apply concepts of ML. In the fortnightly meetings with the supervisors, this subject was discussed, and in syntony, with the company, we decided that defining the structure of intents and the state machine was an important step and that it would make all the sense to dedicate most of the time to this phase because it would be the basis of the implementation. In conclusion, the implementation plan for the second semester was based on the creation of a policy system from its creation to its installation, leaving for future work the monitoring phase where the ML algorithms would be applied.

Chapter 4

System Design

This chapter presents the requirements and system architecture as well as the explanation of each component. In addition to the architecture, the structure of an intent, state diagram and mapping table are presented. Finally, the chapter closes with the use cases.

4.1 Requirements & Architecture

In the first stage, it was necessary to establish which requirements the system should meet. To this end, the table 4.1 was drawn up, using the MoSCoW model [19], which indicates the requirements with the highest priority and which are of interest to the study. To understand the states of this method:

- **Must:** Critical to the system
- **Should:** Essential but not necessary
- **Could:** Desirable but not necessary
- **Won't:** Less critical

Code	Requirement	MoSCow	Who
R1	User must be able to indicate request to the system	MUST	User
R2	The system must recognise the user's request	SHOULD	System
R3	The system should respond to user actions	MUST	System
R4	The system must be able to implement the user's request	SHOULD	System
R5	The system must be able to detect conflicts between different user actions	SHOULD	System
R6	The system must be able to optimise the network	SHOULD	System
R7	The system must be able to negotiate with the user	MUST	System/User
R8	The system must be able to detect anomalies in the network	COULD	System
R9	The system must be capable of alerting the user in the event of a malfunction	COULD	System

Table 4.1: MoSCoW method

The system's architecture seeks to meet the requirements presented to implement high-level policies from network users and monitor these policies and the network, optimising if necessary. Figure 4.1 indicates the different logical components of the system and how they interact.

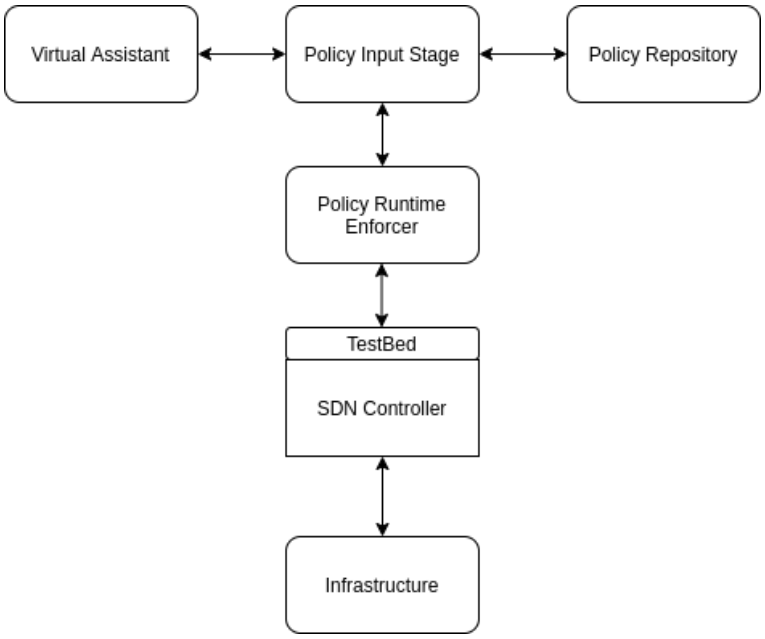

Figure 4.1: System overview

The system architecture is designed to operate in a business context, where the user interacts with the assistant using his equipment (e.g. computer). Therefore, the architecture modules are explained in the following section.

To capture the user's commands, the virtual assistant is the component responsible for maintaining communication between the system and the user. After being activated, the user saves the user's intention and forwards it to the system to validate and implement it. The assistant is in charge of associating the policy with the user's intention and converting the high-level language into a machine language, sending it to the component where it will be analysed.

The policy repository is the component responsible for storing all existing policies in the system as a database.

Responsible for communication between the virtual assistant and the policy repository, the Policy Input Stage is the component in charge of analysing conflicts so that the user's policy does not conflict with some rule already implemented in the network. In case of success, this component sends the rules to be implemented. On the contrary, if there is a conflict, this component initiates negotiation with the user. Furthermore, the thresholds to validate the policies are part of this component, sending to policy runtime enforcer for analysing and drawing conclusions.

The Policy Runtime Enforcer is responsible for constantly monitoring the network and creating the request for installation. In this component, information from the network is collected, analysed, planned, and finally executed. It understands the problem and how to envisage policy implementation, such as operations and resources. An important factor is planning, which plays a critical role in accepting policies and why it will be the element that will govern a possible negotiation with users. In addition to planning, data analysis is responsible for analysing the previous component thresholds to act if a policy fails. Algorithms of ML are applied in this module.

TestBed is an API that is in charge of the communication between the system and the controller SDN. This API receives the installation request from the Policy Runtime En-

forcer and tries to install it in the controller. In this component, all the installation logic is set, knowing which requests the controller supports or not. Besides installation requests, there are data collection requests that will be useful to apply ML algorithms and calculate interesting variables for the study.

SDN controller is in charge of obtaining both the data from the network and implementing any rules. In addition, the controller communicates with the infrastructure, obtaining the necessary information to be used for the system to make the changes it wishes.

4.2 Internal system operation

As in section 4.1, we can state that the system consists of two different but connected components. The first one is responsible for implementing high-level language policies, and the second one monitors the network to ensure that it is within the defined parameters. However, one component completes the other so that the system can function.

In the following figures, two interaction diagrams are presented to understand how the system operates internally.

Figure 4.2 the user interaction with the system is presented in the form of a diagram generically, adapted to any case of use.

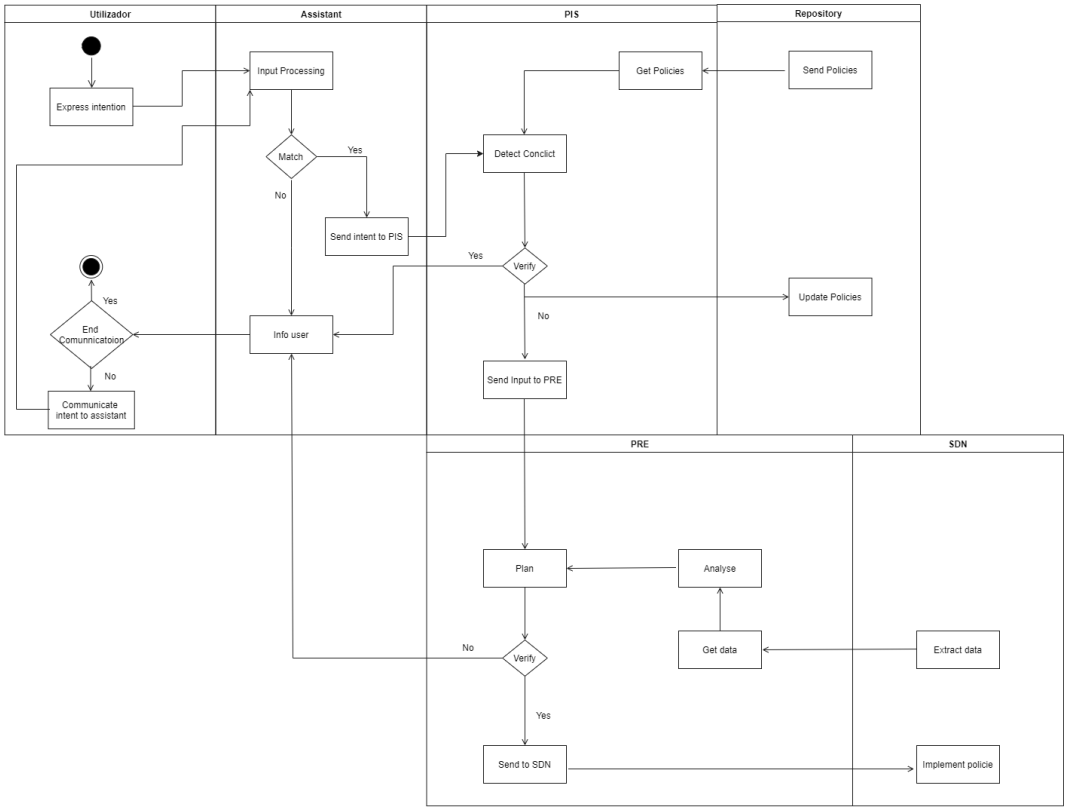

Figure 4.2: Internal system operation flowchart in a generic interaction - implementation of high level policies

The user activates the system when he feels the need to apply policies on the network. The virtual assistant processes the user's input, checking that the expressed intention is valid and ready to be sent. If there is any problem in the match of the intention informs the user, there are two possibilities: end of interaction or need for further action by the

user. It converts it into the desired format and sends it to the input stage policies if it is valid. In this component, the policy is decomposed to analyse the conflict between policies to prevent the policy from being implemented and disagree with any existing. In case of conflict, the system alerts the assistant, alerting the user, as mentioned above. If there is no conflict, the policy is sent to its execution time, hung where it is analysed to apply this new rule. Information about the network is extracted and analysed. If an infrastructure problem prevents the new rule from taking action, the user is alerted again, as in previous cases. If there is no problem, the new rule is sent to the controller to be implemented on the network.

In Figure 4.3, a flow chart is shown where you can see the system monitoring process over the network.

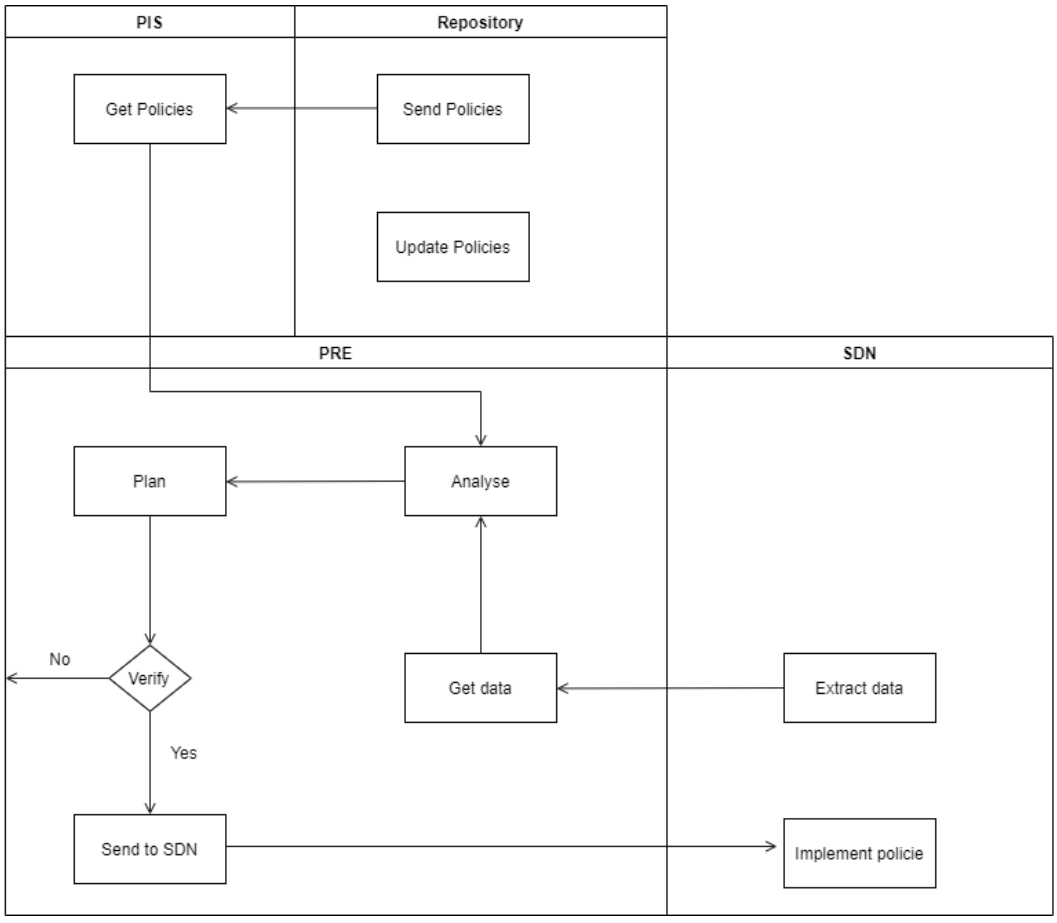

Figure 4.3: Internal system operation flowchart in a generic interaction - monitoring

The system captures data from the network. This data is analysed using the policy repository thresholds to see if the rules are being followed. Algorithms of ML are applied to classify traffic and optimise functions on the network. Then it is planned how the previous network decisions will be implemented so that the network complies with what has been implemented. If we see no problem in the implementation, we arrive at success, and it is implemented.

4.3 Intent Structure

The structure of intents represents all the information coming from the user structured so that the system can interpret. The system can receive several requests from several users, so it is necessary to create a structure that can cover as much varied information as possible. For this, the structure shown in Figure 4.4 was created.

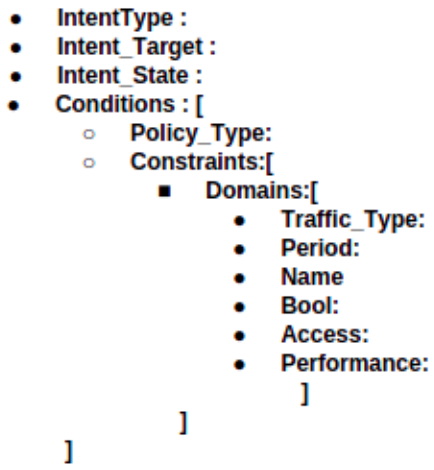

Figure 4.4: Intent Structure

For each request, is assigned a type (example: indicates whether the request is to set priorities or create services), a target (example: what type of network) and a state (based on the state machine). Besides these three points, policies are defined according to the user's wishes, and for each policy, we can have restrictions that can indicate performance values, type of traffic, who accesses the network, among others that are presented in Figure 4.4.

4.4 Intent State Machine

A state machine was created to understand how the system works and covers the possible states reached by requests made by the user. This state machine resulted from a more rigorous investigation to which much of the time of this study was devoted since it defines how we will approach the problem and deal with it. In Figure 4.5 we can visualise the result of the state machine.

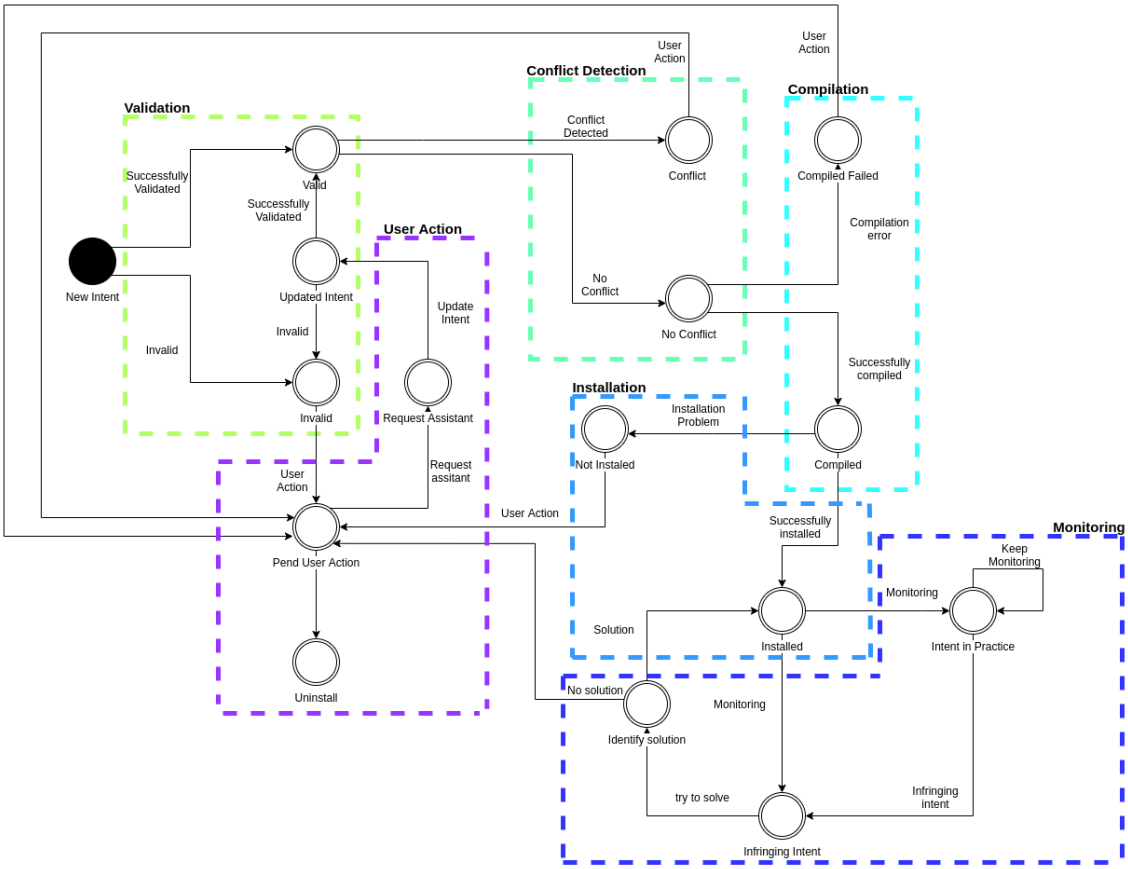

Figure 4.5: Intent State Machine

As we can see, the state machine is divided into six sections. After the user requests in the first phase, the validation phase verifies that the request contains all the necessary information to implement the desired action. If the user has forgotten to mention necessary information for the request to be implemented or if the information he has provided is wrong, this phase assigns an invalid status, which will require user interaction. On the other hand, if the user provides all the necessary information for the request to proceed, it assigns a valid status and continues to the next phase.

The conflict phase is the second checkpoint of this state machine. Here, the already validated attempts are subject to a comparison with the requests already implemented and stored in the database to conclude conflicts. If the request is redundant, or if the new request consists of information contrary to that already implemented, the conflict phase assigns a conflict state which will be resolved with the help of the user. If the information does not conflict, we move on to the compilation section.

When the request reaches the compilation phase, the system tries to convert into rules the policies that the user wants to implement, i.e., the system converts the high-level language into a language that can be interpreted by the controller so that it is then possible to install them. Sometimes the desired information may not be supported by the controller, or the desired operations may not be operational, and in this case, the compilation phase gives a compilation error that can be solved with the help of the user. If we can get the rules to be installed, we move on to the installation phase.

Once the rules are obtained, the next step is to install them on the controller. If the objectives are supported by the controller and installed, we move on to the monitoring

phase. If the goals are not achievable because they may be offline or nonexistent, the user is alerted that the application was not installed.

In the final phase, monitoring consists of a cycle that constantly checks if the existing requests in the database are being fulfilled or if any violation has occurred. If any non-compliance occurs, the system, through intelligent algorithms, tries to identify how to solve the problem without human intervention automatically. To make this possible, the area of ML enters this phase, whereby capturing data from the network, the system tries to identify patterns in order to validate existing policies and intervene if necessary. If the intelligent algorithms do not identify a solution that corrects the problem, the user is alerted.

In Figure 4.6 we can visualise how the cycle of an event works from creation to installation.

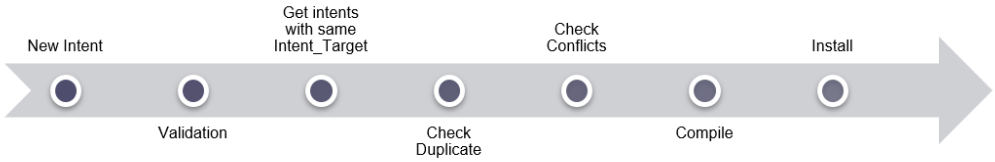

Figure 4.6: Installation process

4.5 Mapping table

To create a rule about any device or service on the network, it needs to know which IP is assigned on the network. However, it would be inappropriate for the user to know all the IPs and still have to indicate them to the system every time he went to the system to implement something. Having this problem, creating a mapping table that assigns names to all the hosts on the network was necessary. An example of this table can be seen in Figure 4.7.

Name	IP
one	192.168.2.1
four	192.168.2.4

Figure 4.7: Mapping table

With the mapping done, the user does not have to know the IP range of the hosts to which they want to apply rules but rather the names of the respective hosts, facilitating how they interact with the assistant.

4.6 Use cases

In this section, the use cases implemented in the system are introduced. The use cases that are presented were designed in order to explore the possible scenarios presented in section 4.4, illustrating the most common cases in a business environment. Integrating virtual assistants with network management systems enables network functionalities without the user having specific knowledge in the area.

In the figure 4.8 we can visualise the use cases diagram that responds to the requirements mentioned in section 4.1.

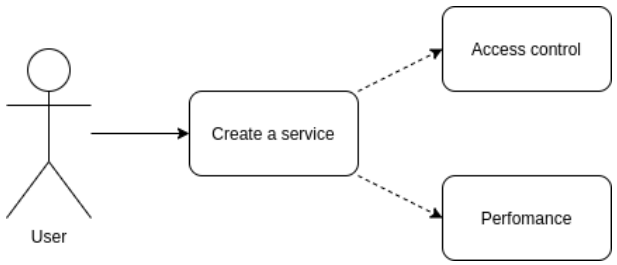

Figure 4.8: Use cases diagram

The use case presented in the diagram is divided into two sub-use cases that control the access and performance of the new service. To better understand this, the 4.2 table is presented.

Name	Create a new service
Description	The user is able to request the creation of a new service through the virtual assistant. The latter will check if all necessary information to describe the service is present and if not it will request it from the user. Afterwards, it will try to configure the new service on the network
Actors	-User -Virtual Assistant -SDN Controller -Network elements -Devices connected to the network
Preconditions	-There is a database that identifies the different devices connected to the network -The network is operating under normal conditions -There may be services already running(optional) -The virtual assistant is deployed and available -The SDN controller is already running and connected to the network -There are already multiple devices connected to the various network switches
Postcondition	The service is accepted and is configured on the network
Step-by-Step Sequence	1. The user requests the creation of a new service 2. The virtual assistant asks the user to provide a name for the service 3. The user provide the service name 4. The virtual assistant asks the user who can access the service 5. The user provide the hosts that can access 6. The virtual assistant asks the user if he wants to define a performance for the service 7. The user provides the performance that the service will support 8. The virtual assistant asks the user if he wants the service to have internet access 9. The user indicates if he wants the service to access the Internet 10. The virtual assistant acknowledges that if has the necessary information to specify the service 11. The virtual assistant performs conflict analysis 12. The virtual assistant starts the compilation process 13. The virtual assistant creates the new service 14. The order successfully completes all the necessary steps from its creation to installation. 15. The virtual assistant confirms to the user the creation of the new service
Alternative scenarios	10. If the information necessary for the implementation of the service is not available, the user is notified 11. If there are conflicts between new requests and existing requests in the database, the user is notified 12. If the compilation process fails, the user is alerted 13. If the virtual assistant fails to create the new service, the user is alerted.

Table 4.2: Use case description

The use case presented 4.2, aims to create new network services. This insertion process touches on several essential points of this project, from the negotiation process of the service name to the insertion in the network without jeopardising existing services that have performance dependencies on the network itself.

For a better interpretation of the use case 4.2, a dialogue between the user and the virtual assistant is presented. The user, through voice commands, can ask the virtual assistant to

create a new service by pronouncing the following sentence:

- User: I would like to create a service

Once the virtual assistant captures this command, the user is asked what the name of the new service to be inserted(negotiation) is, to which the user can respond as follows:

- Assistant: Sure. What is the name of the service?
- User: Name it s2
- Assistant: Please name the machines that will have access to the service
- User: Provides access to hosts 3 and 6
- Assistant: Regarding internet access, does the service need internet access?
- User: No, thanks
- Assistant: One last question, Do you want to define the performance of the service?
- User: yes please, 4000

4.7 Conclusion

In this chapter, we start by describing the system requirements that gave rise to the presented architecture. Then, this architecture is presented by six components that interact with each other. We separated the system logic into two phases based on this architecture: high-level language policies and monitoring. After creating the logic, we describe the structure of intent as well as its life cycle. These were two stages to which time was dedicated since they directly influenced the system's proper functioning. Next, a mapping table was presented to make life easier for the user when interacting with the virtual assistant. Finally, we conclude with the use case, service creation, which involves two sub-use cases (access control and performance). It was decided to assume as a single use-case since separate would not make much sense.

Chapter 5

Framework

This chapter will explain which components allow the system to function, choosing technologies for the virtual assistant, databases and controller. The second point is how the state diagrams were implemented, ending with the implemented use cases.

5.1 Technology selection

The selection of the technologies was an essential step in the development of this project because it allowed us to dictate which functions we could use to have a complete system.

Starting with the virtual assistants studied in section 2.3 and comparing with the interests of the project, as the features that the assistant should have (communicate with the user through voice command, skills handling, conversion from voice command to the appropriate language), we concluded that the Mycroft AI [20] was the ideal choice and answered most of our problems. This assistant allows us to create new skills quickly and, with the help of internal software, Adapt Intent Parser [1], get the information we need in a JSON data structure, to formulate the user request, sending it for further analysis and implementation.

Once the virtual assistant was chosen, we needed to decide where to store the requests coming from the user. For this, the Apache CouchDB [8] tool was chosen, a non-relational database that allows working with JSON files, which turns out to be an added value since this is the format generated by the virtual assistant. Furthermore, Apache CouchDB is a technology that allows you to store, edit and remove documents through REST requests in a simple way. This way, whenever the system validates a request, it is stored in the database.

To finish the section on the choice of technologies, we have the SDN controller. For this technology, several solutions were analysed in the 2.1 section. Finally, based on the problem of this dissertation and analysing the support of the analysed controllers, the controller ODL-Carbon version was chosen. This controller supports several interest requests to the study, such as topology, trace hosts, flow management, and network statistics. Since it answers most of the problems, we proceed with this controller.

Finishing the choice of technologies, we start the explanation of the components shown in the state diagram 4.5

5.2 Validation stage

As mentioned in section 4.1, responsible for the communication between the virtual assistant, policy repository and the installation component, the Policy Input Stage was one of the first components to be created. This module is in charge of receiving the user's request, starting by validating it. The first phase of the state diagram 4.5 begins the validation. As we can see in Figure 5.1, an attempt to reach the system can reach two states: validated or invalid.

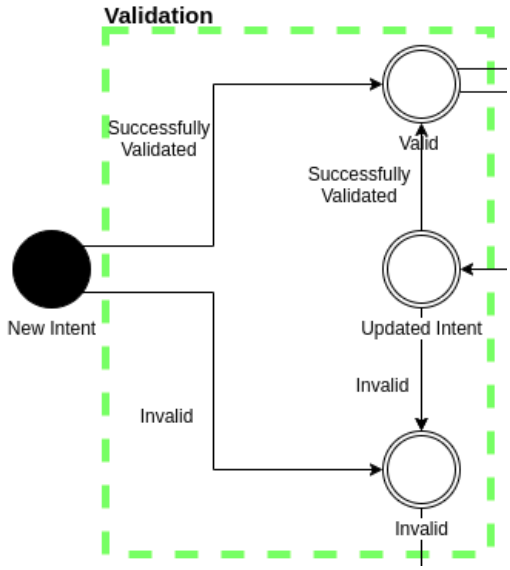

Figure 5.1: Validation stage

To validate, the system verifies that all the information required to implement the request is available and feasible. To better understand when an intent reaches an invalid state, an example is shown in Figure 5.2.

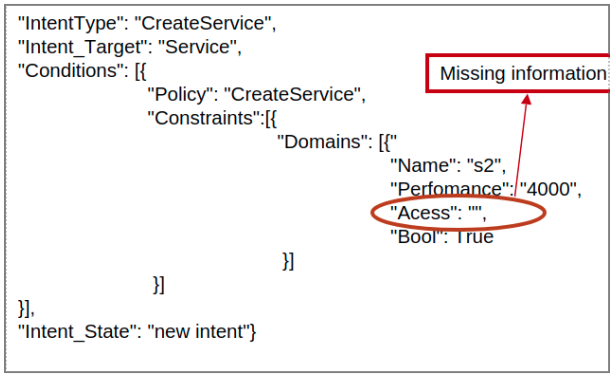

Figure 5.2: Invalid Intent

For this validation to be possible, a set of minimum requirements (mandatory fields) is defined for each type of policy, which will be checked whenever they arrive at the system. So, for example, for this specific case 5.2, to create a service, it is necessary to validate the service name, who can access it, if there is Internet access and if a performance has been defined. Once the validated state is reached, the attempt moves on to the conflict phase.

5.3 Conflict stage

In the context of the API- Policy Input Stage, after the validation process and the assignment of a valid state to the intent, the conflict stage starts. As we can visualise in Figure 5.3, and intent when entering this stage can reach two states: conflict or no conflict.

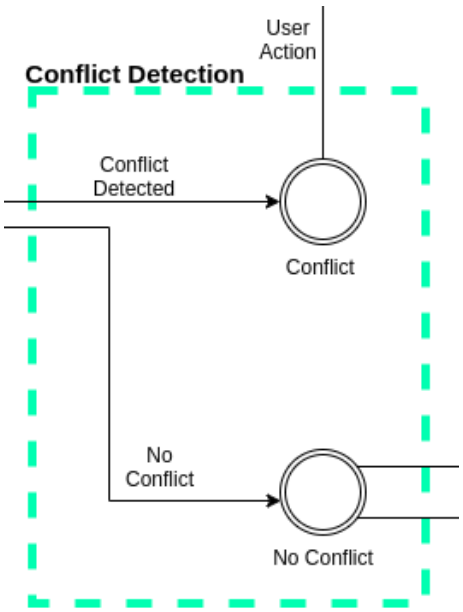

Figure 5.3: Conflict stage

To assign these states, we need to get all the policies stored in the database related to the new policy that arrives in the system. To do so, it was necessary to create a function that, based on the "IntentType", see section 4.3, returns all the policies already implemented that are of the same type as the new one in the system. This starts the process of checking and comparing fields so that no conflicts occur.

To better understand how conflicts are detected, an example of a conflict for the use case indicated in 4.6 is presented in Figure 5.4, where the user has to be alerted about the conflict in question.

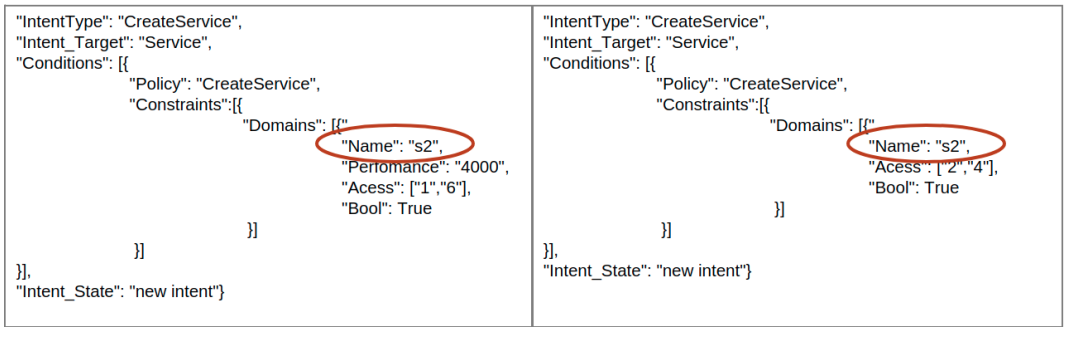

Figure 5.4: Conflict Intent

The conflict presented is based on the idea that no two services can exist with the same name. Therefore, when the user inserts a new service, a check is performed, which consists of obtaining all the services already implemented to verify if the new service's name already exists. In short, the conflict phase compares the fields of the new policy with the fields of policies of the same type in order to check if there is any disagreement regarding, for

instance, redundant names or services sharing the same hosts. With these checks, we will not have conflicting rules in the system.

Once the validation and conflict phase is over, the application goes on to compilation and installation.

5.4 Compilation stage

As mentioned in section 4.1, the API- Policy Runtime Enforcer is responsible for creating requests to install new policies on the network, of which the compilation phase is part. As soon as the intent is assigned the valid and conflict-free status, it enters the compilation phase where the high-level language, previously defined through the Intent structure 4.4, is converted into a language interpretable by the controller. As we can see in Figure 5.5, an intent entering the compilation phase can reach two states: compiled or failed compilation.

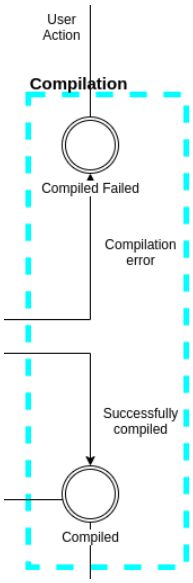

Figure 5.5: Compilation stage

If the controller supports the desired information, i.e. some commands support the user's request, it proceeds for installation. However, sometimes, the controller may not support the request, or the operations may not be available, which leads to a compile failure state. For example, in Figure 5.6, we visualise the compilation process for a request to create services with bandwidth limit and Internet access.

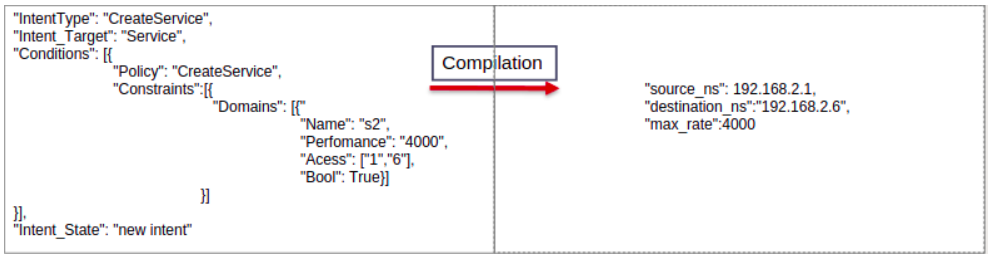

Figure 5.6: Compilation of priority Intent

The structure presented on the left side of Figure 5.6, during the compilation phase, is

divided into several requests that will achieve the intent's goal. One of these requests is the definition of the bandwidth limit supported by the service, in this case, 4000 Mbit sec between hosts 1 and 6.

5.5 Installation stage

As mentioned in the 4.1 section, responsible for the communication between the system and the SDN controller, the installation phase is part of the API-TestBed. This component dictates which resources the system supports, i.e. which applications can or cannot install. The information that the user wants to insert in the network arrives from the previous API, where the data is already in a format supported by the controller, and here the installation request is made to the ODL. As we can see in Figure 5.7, an intent when entering the installation stage can reach two states: installed or not installed.

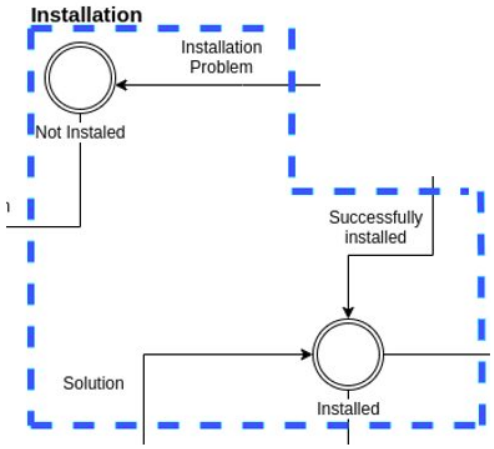

Figure 5.7: Installation stage

As for the requests supported by this phase, these are:

- Create a new Switch
- Delete a Switch
- Connect the two chosen switch between each other
- Create the chosen number of Network Namespace and connect it to the Switch
- Create and add a flow
- Delete the chosen flow throw the id
- Get all topology from the system
- Get information from all ports in the system
- Create QoS
- Delete QoS

To implement these applications, it was necessary to work with technologies such as Open vSwitch and Openflow. The former is a switch platform that supports standard management interfaces and opens up routing functions for programmatic extension and control. There are components in the Open vSwitch (OVS) distribution that allow users to create a virtual network with multiple switches and manage traffic between them using OpenFlow. In addition, OpenFlow allows network controllers to determine the path of network packets through a network of switches. In this way, it is possible to implement the use cases presented in the 4.6 section.

5.6 Monitoring stage

The monitoring phase checks if the existing requests in the database (installed requests) are being fulfilled or an error. As we can see in Figure 5.8, an intent entering the monitoring phase can reach three states: intent in practice- the intent is being fulfilled; infringing intent- the intent was violated; identify a solution- the system tries to find a solution so that the intent can work properly again.

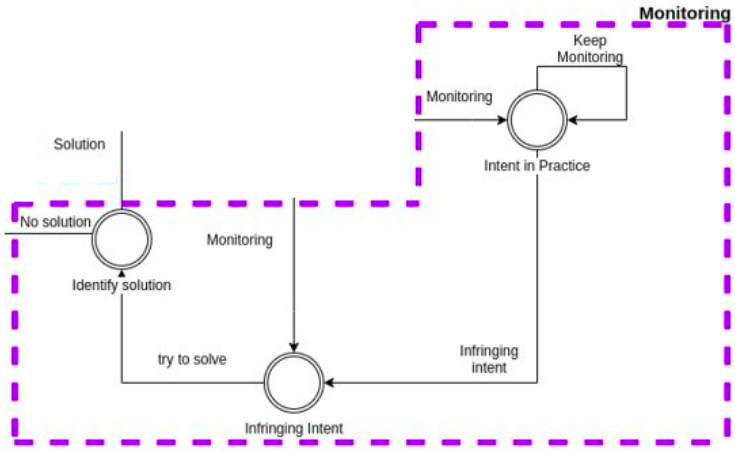

Figure 5.8: Monitoring stage

This phase was not developed due to the work that was arising and the fact of having to do further research on the intents structure 4.4 and the state diagram 4.5. Moreover, in addition to this investigation, the validation, conflict detection, compilation and installation process ended up taking up much of the time, taking away what I had planned to devote to this phase. However, the research presented in the literature section 2.2.2 will serve as a basis (along with future research), for the implementation of this component, in order to detect anomalies in the network and solve them, with the help of intelligent algorithms, without user intervention.

5.7 User action

The last component of the state diagram is the user action. In this phase, the virtual assistant, Mycroft, communicates with the user for any error in the system, from validation failures, conflicts, compilation, installation and monitoring. As we can see in Figure 5.9, for this component, there are three different states: Pend User Action- a state where user

intervention is required; Request Assistant- the assistant intervenes to collect information; Uninstall- uninstalls intent.

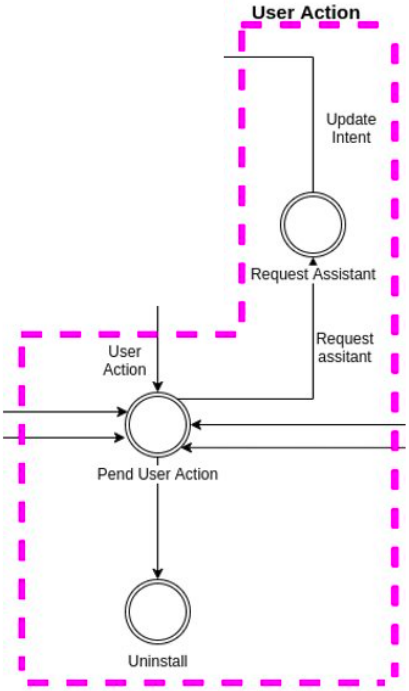

Figure 5.9: User action

The virtual assistant receives several requests from the system (alerts or information about the state of the attempt) to inform the user and decide what actions to take.

With the state diagram defined and implemented, it was possible to create a scenario that proves that the user can ask an intelligent assistant to control the network.

5.8 Use cases implementation

In this section, the whole process of the use cases mentioned in section 4.6 is presented. As mentioned in the mentioned section, the presented use case, service creation, involves other sub-use cases like network access control, performance definition and Internet access. Figure 5.10, depicts the process involving this request between the user, the assistant and the system.

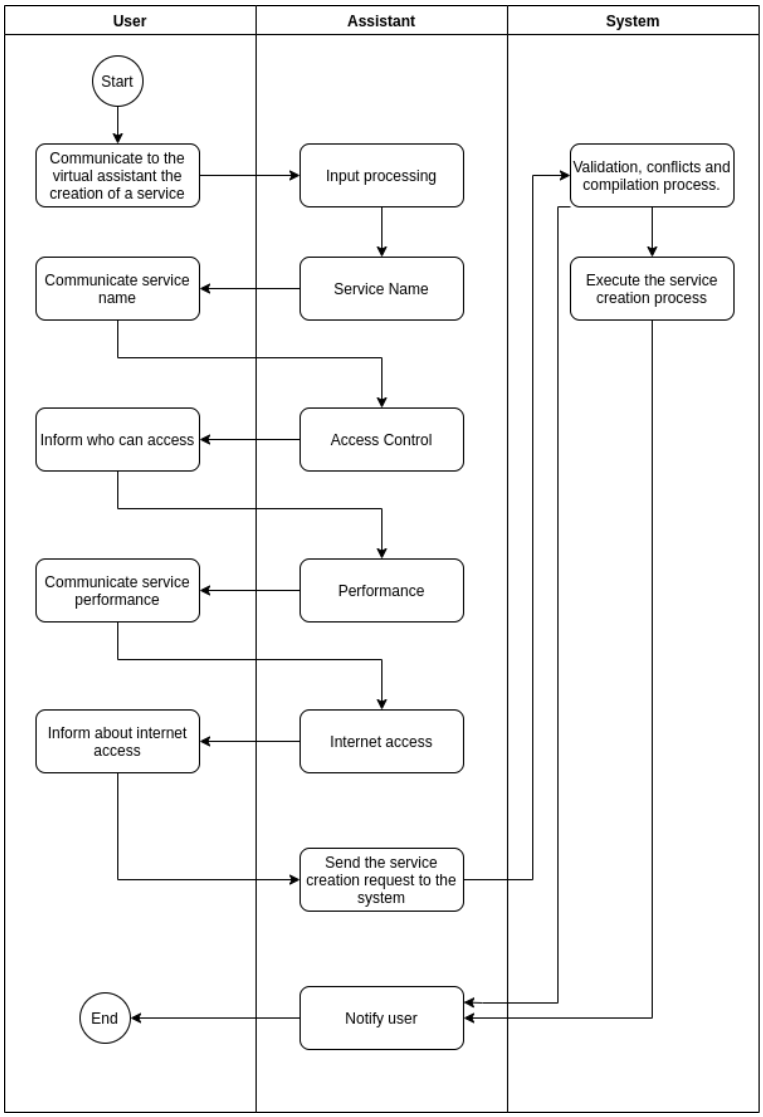

Figure 5.10: Use case - implementation

After pronouncing the service creation request on the assistant, a negotiation process starts between the two entities to obtain all the necessary information for the service creation. This information is summarised in the name of the service, who can access it if there is a bandwidth limit, and finally if there is access to the Internet. After the user indicates all the information, the assistant starts the service creation process by sending the request to the system.

As mentioned in section 4.4, the first step of the system is the validation phase, where the request goes through a set of rules that validate the completion of the fields in the request. For this use case, the validation phase consists of checking the information about the service name and who can access it, as these are the minimum mandatory requirements for creating a service. To do so, it validates if the name field is different from null and associates the hosts that can access the service with the Internet Protocol (IP)s, present in the mapping table 4.5, in order to check if they exist. Besides checking these fields, it validates the internet access and performance variables. If any problem occurs, such as missing or wrong information, like hosts that do not exist, the user is notified.

Once validated, the conflict phase begins, where a request is made to the database to check

if there are services on the network. If yes, it is necessary to compare the existing services with the new ones so that no services have the same name. Besides that, it is analysed if the hosts associated with the service are involved in other services to alert the user. Finally, if the service name already exists, a conflict is generated that forces the user to create another service with a different name.

When we enter the compilation phase, we get all the information from the user to know which requests will be created. First, a topology request is made to API-TestBed, which returns all the information of hosts and switches involved in the network. In Figure 5.11 we can view an excerpt of the topology request response.


```
{
    "network-topology": {
        "topology": [
            {
                "topology-id": "flow:1",
                "node": [
                    {
                        "node-id": "openflow:107953924712512",
                        "opendaylight-topology-inventory:inventory-node-ref": "/opendaylight-inventory:nodes/
                            opendaylight-inventory:node[opendaylight-inventory:id='openflow:107953924712512']",
                        "termination-point": [
                            {
                                "tp-id": "openflow:107953924712512:13",
                                "opendaylight-topology-inventory:inventory-node-connector-ref": "/
                                    opendaylight-inventory:nodes/opendaylight-inventory:node
                                    [opendaylight-inventory:id='openflow:107953924712512']/
                                    opendaylight-inventory:node-connector
                                    [opendaylight-inventory:id='openflow:107953924712512:13']"
                            },
```


Figure 5.11: Excerpt from the topology request

Once this information is available, a request is made to API-TestBed with information about the network statistics. This request returns information about the Throughput between all the links that make up the network every 15 seconds. In Figure 5.12 we can check which statistics are returned by the API.


```
"openflow:34076490910029:107": {
  "BytesReceived": 0,
  "BytesTransmitted": 1026,
  "Name": "b3-eth7",
  "PacketsReceived": 0,
  "PacketsTransmitted": 9,
  "PortConnectionDuration": 722,
  "PortFeature": "ten-gb-fd copper",
  "PortNumber": "107",
  "PortSpeed": "10.0 GB/s",
  "Throughput": 0.00013680000000000002
},
```


Figure 5.12: Statistics captured by API-Testbed

In turn, after knowing the value of this variable for all links, we create a graph (based on the topology information) with all nodes and assign as the weight between nodes the variable Throughput. This way, we have a complete graph where the nodes represent the network hosts and switches and the vertices the Throughput between them. An example of this graph is shown in Figure 5.13, where the hosts are represented by blue and switches by yellow.

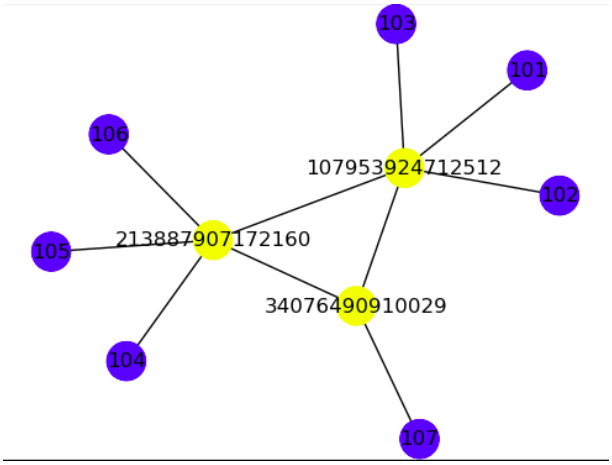

Figure 5.13: Topology graph

Based on this graph, we apply the Dijkstra algorithm in order to find the shortest path between nodes based on the weights assigned to the vertices [9]. This algorithm will return the best paths between hosts that can access the service to insert it without jeopardising existing services that have performance dependencies on the network itself. Figure 5.14, represents the path returned by this algorithm between host 103 and 107.

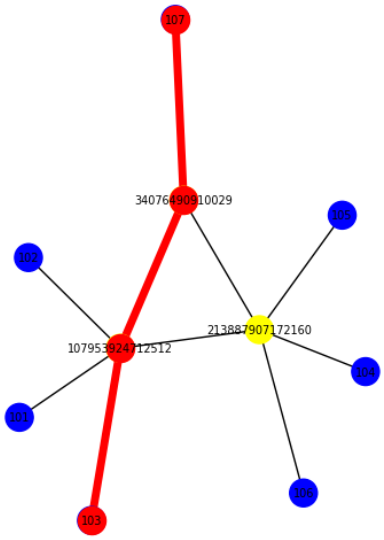

Figure 5.14: Topology graph - Dijkstra algorithm

Once we know the service route, we verify if there will be Internet access and, if so, we obtain the best path between the hosts involved in the service and the one that allows Internet access. Finally, we verify if the user has defined some performance for the service. If so, the request is created between the hosts present in the service with the maximum limit set by the user. If no performance was provided, then the service is not limited, being able to enjoy the maximum network capacity. Having all compiled requests, these are sent to the API-TestBed, where they are installed. In Figure 5.15 you can see the information needed to add a flow.

```
{"Switch":"openflow:196100289772364",
"in-port":"101",
"out-port":"12",
"ipv4-src":"192.168.2.1",
"ipv4-dst":"192.168.2.2"}
```


Figure 5.15: Request to add flows

With this request, the service is created between the hosts that the user wants to access. For example, in the case of an Internet connection, the connection between the hosts involved in the service and the host associated with the Internet is created.

As for performance, the request is based on the following information 5.16.


```
{"source_ns":"b1_NS1","destination_ns":"b2_NS2", "max_rate":"4000"}
```


Figure 5.16: Request to QoS

Knowing the limit set by the user, the service capacity is limited by sending the presented information to the controller.

5.9 Conclusion

The selected technologies were the first focus of this chapter. This selection resulted in technologies such as Mycroft, ODL and Apache CouchDB. Next, it was presented how the implementation of the validation, conflicts, compilation, installation, monitoring and user action states were faced. For each phase, the possible states were presented, and an example was given. Finally, the use case was described with the help of a diagram that explained the interaction between the system components, and it was explained for each state machine phase how the system approached the user's intent, with the help of some images to contextualize.

Chapter 6

Experimentation

This chapter aims to present the tests chosen to evaluate the system performance. The elaboration of these tests considers the implemented functionalities of the system, the user being able to communicate with the assistant in several ways. In each test instance, the user was placed in ideal conditions to perform it, and only the necessary information regarding the interaction with the assistant was transmitted to create the desired environment for the system evaluation. This test evaluates the functionality of creating a service. The test consists in putting the user in a situation that leads him to create a service. For that, he needs to interact with the virtual assistant.

6.1 Creation of the test environment

To apply this test, an environment was created that allows the system to control the network according to the user's wishes. This environment was generated with the help of the software OVS, which through the commands shown in Figure 6.1, allowed the creation of the hosts and switches on which the new rules will fall.

Create Switch
curl --header "Content-Type: application/json" --request POST --data '{"Switchs":"1", "Name":"b1", "GW":"192.168.2.156"}' http://127.0.0.1:5000/createSW

Connect Switches
curl --header "Content-Type: application/json" --request POST --data '{"NameSwitch1":"b1", "NameSwitch2":"b2"}' http://127.0.0.1:5000/connSW

Create Hosts
curl --header "Content-Type: application/json" --request POST --data '{"Switch":"b1","Hosts":"3","Flag":"2"}' http://127.0.0.1:5000/create

Figure 6.1: Setting up the environment

Three switches were created, connected and to which three Hosts were assigned to Switch 1 and Switch 2, and a Host to Switch 3 (representing the Internet access). Thus, the environment was as shown in Figure 6.2.

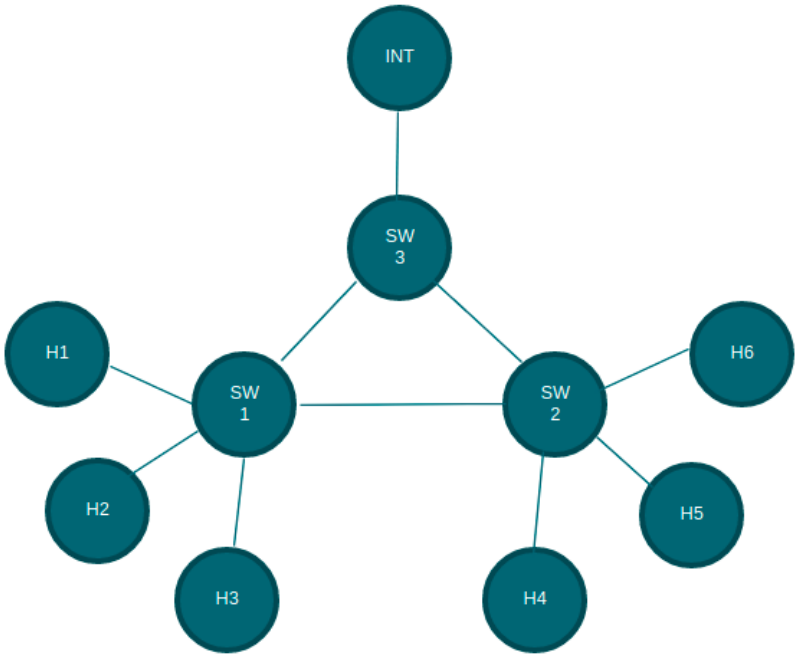

Figure 6.2: Testing environment

Once created, there are no active services on the network. To prove this, a test is performed, with the help of the iperf tool, between Hosts 1 and 6 to prove that there are no active services. Figure 6.3, shows the test between these two Hosts.

```
sudo ip netns exec b2_NS6 iperf3 -c 192.168.2.1 -p 4001 --cport 4006
iperf3: error - unable to connect to server: No route to host
```

Figure 6.3: Connection test between Host 1 and 6

Running the server on Host 1, we try to connect the client Host 6 and iperf does not know the route, as expected. This fact extends to all the Hosts involved in the network. With this, the user can communicate to the virtual assistant the desire to create a service between Host 1 and Host 6.

6.2 Create a service

To demonstrate the communication between the user and the virtual assistant, the dialogues are presented in text form, where the blue sentences represent the user's voice input and the yellow sentences the virtual assistant's voice output. For example, figure 6.4 shows two examples of how the user can ask Mycroft to create a service.

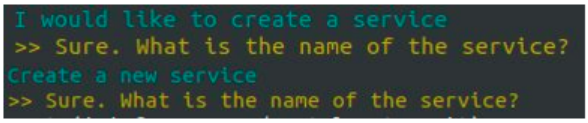

Figure 6.4: Test-Create a service

Once requested to create the service, the virtual assistant asks the name of the service to be created. The user can answer in several ways, as we can see in Figure 6.5.

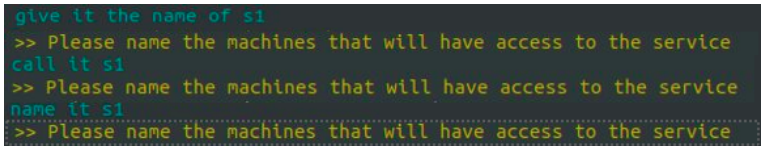

Figure 6.5: Test-Name a service

After providing the service name (s1), the user is asked who can access it. In order to facilitate the interaction with the user, using the mapping table 4.5, the user can indicate through the numbers assigned to the network hosts who can access it. For example, figure 6.6 shows some ways of indicating who can access the service.

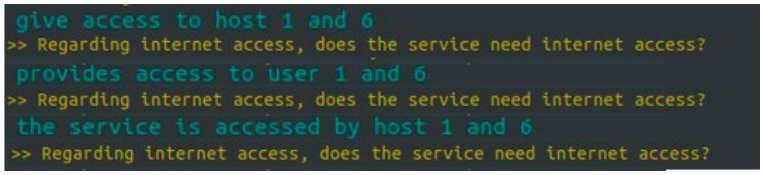

Figure 6.6: Test-Access a service

In this way, the user indicates the users (User 1 and User 6) without needing to know the IP, which will later be assigned to the respective users. The next question addresses Internet access, which can be seen in Figure 6.7.

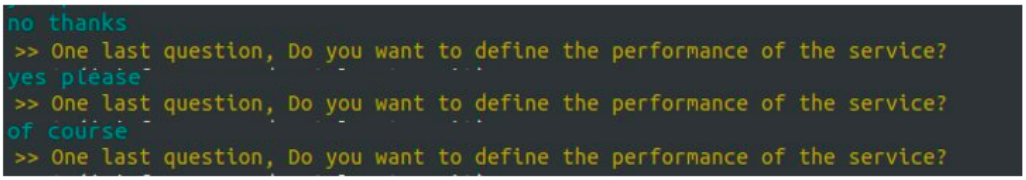

Figure 6.7: Test-Internet access in service

Once the user indicates whether or not they want the service to access the Internet, the virtual assistant asks if they wish to indicate the performance of the service. Again, the user can speak in various ways to the assistant, as can be seen in Figure 6.8

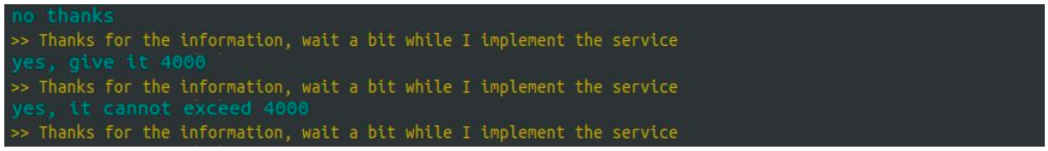

Figure 6.8: Test-Performance in service

Once the request is concluded, it goes to validation (As shown in section 5.2), where the parameters for creating the service are checked. To test the validation, two examples are presented. In the first, the user does not indicate which hosts access the service, answering the remaining questions of the virtual assistant correctly. Thus, the assistant, when validating the parameters, verifies the absence of the hosts and alerts the user to the lack of information, see Figure 6.9.

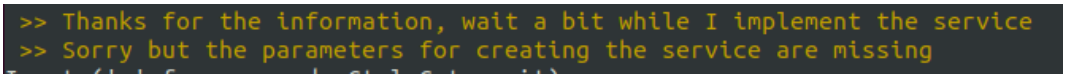

Figure 6.9: Test-Missing parameters

Upon being notified, the user will have to redo the service creation request. For the second test, the user indicates hosts not present in the network, either because they do not exist or are inactive. The virtual assistant, when trying to match the IPs with the names, detects that these do not exist, notifying the user, as shown in Figure 6.10.

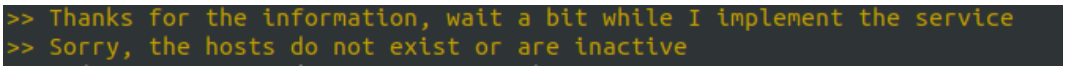

Figure 6.10: Test-Unknown users

Once again, when the user is informed that the service users do not exist, he will have to recreate the request. If the request is validated, it enters the conflict phase (As shown in section 5.3), where two conflict examples are presented for this test. The first one is when the user creates a service with the same name as a service already implemented. For this case, the assistant verifies all the existing services in the network and looks if there is any with the same name as the new service. If so, it alerts the user of the error, as we can see in Figure 6.11.

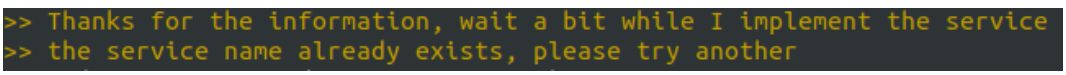

Figure 6.11: Test-Service name already exists

When the virtual assistant notifies the user about the error, the user will reformulate the request. The second conflict serves more as a warning than an error, that is, if the user, when creating the service, indicates some host that is present in another service, Mycroft alerts to that fact, without the request ceasing to be implemented, see Figure 6.12.

```
>> Thanks for the information, wait a bit while I implement the service
>> The hosts involved are associated to other services
```

Figure 6.12: Test-Users involved in other services

Once validated and without conflicts, the request goes to the build phase (As shown in section 5.4). In this phase, there are several requests made to both the API-TestBed and the database in order to convert into rules the policies coming from the user. If any of the requests fails, the compilation phase fails, and there is the need to alert the user, as we can see in Figure 6.13.

```
>> Thanks for the information, wait a bit while I implement the service
>> When trying to compile your request, an error occurred. Please try again
```

Figure 6.13: Test-Compilation error

When we successfully get the rules compiled, we move on to the installation phase. In this phase, the rules are sent to API-TestBed, which in turn communicates with ODL to install them. After they are installed, they are stored in the CouchDB database, and the user is notified that the application has been successfully installed. The figure 6.14 shows the case of successfully installed (As shown in section 5.5).

```
>> Thanks for the information, wait a bit while I implement the service
>> Service created successfully
```

Figure 6.14: Test-Request successfully installed

To analyse whether the service was created with the restrictions made by the user, let us assume that answered the questions asked by Mycroft as follows:

- Service Name: s1
- Access: host 1 and 6
- Internet Access: Yes
- Performance: yes, 4000

By interpreting this information and bypassing all the processes that constitute the state machine 4.4, and were talking about earlier, the system creates the service. To test whether the service has been created between hosts 1 and 6, with Internet access and bandwidth limit set, we again use the iperf tool, in the same context mentioned above 6.3. Figure 6.15 shows the result of the service created between Host 1 and Host 6, with the bandwidth limit close to the user-defined value, 4 Mbits/sec.

```
Connecting to host 192.168.2.1, port 4001
[  5] local 192.168.2.6 port 4006 connected to 192.168.2.1 port 4001
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  1.62 MBytes  13.6 Mbits/sec  115   5.66 KBytes
[  5]   1.00-2.00   sec   573 KBytes  4.69 Mbits/sec   77   4.24 KBytes
[  5]   2.00-3.00   sec   445 KBytes  3.65 Mbits/sec   63   7.07 KBytes
[  5]   3.00-4.00   sec   445 KBytes  3.65 Mbits/sec   74   18.4 KBytes
[  5]   4.00-5.00   sec   509 KBytes  4.17 Mbits/sec   74   2.83 KBytes
[  5]   5.00-6.00   sec   382 KBytes  3.13 Mbits/sec   60   4.24 KBytes
[  5]   6.00-7.00   sec   636 KBytes  5.21 Mbits/sec   83   5.66 KBytes
[  5]   7.00-8.00   sec   318 KBytes  2.61 Mbits/sec   69   2.83 KBytes
[  5]   8.00-9.00   sec   573 KBytes  4.69 Mbits/sec   73   1.41 KBytes
[  5]   9.00-10.00  sec   509 KBytes  4.17 Mbits/sec   72   5.66 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  5.91 MBytes  4.96 Mbits/sec  760             sender
[  5]   0.00-10.00  sec  5.57 MBytes  4.68 Mbits/sec                  receiver
```

Figure 6.15: Connection test between Host 1 and 6(after service created)

Looking at the average Bitrate, we realise that the bandwidth value is not the same as the one set by the user but slightly close. However, if we work with values higher than Mbits/sec, the difference is no longer remarkable. If another host tries to access this service, it will not recognise the route. To check Internet access, we open the iperf server on Host 7 (representing Internet access) and the client on Host 1 or Host 6, see Figure 6.16.

```
Connecting to host 192.168.2.7, port 4007
[  5] local 192.168.2.6 port 4006 connected to 192.168.2.7 port 4007
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  1.87 MBytes  15.7 Mbits/sec  113   5.66 KBytes
[  5]   1.00-2.00   sec   382 KBytes  3.13 Mbits/sec   73   18.4 KBytes
[  5]   2.00-3.00   sec   445 KBytes  3.65 Mbits/sec   93   4.24 KBytes
[  5]   3.00-4.00   sec   445 KBytes  3.65 Mbits/sec   63   5.66 KBytes
[  5]   4.00-5.00   sec   636 KBytes  5.21 Mbits/sec   70   19.8 KBytes
[  5]   5.00-6.00   sec   509 KBytes  4.17 Mbits/sec   71   4.24 KBytes
[  5]   6.00-7.00   sec   445 KBytes  3.65 Mbits/sec   78   5.66 KBytes
[  5]   7.00-8.00   sec   445 KBytes  3.65 Mbits/sec   63   4.24 KBytes
[  5]   8.00-9.00   sec   445 KBytes  3.65 Mbits/sec   76   2.83 KBytes
[  5]   9.00-10.00  sec   573 KBytes  4.69 Mbits/sec   66   5.66 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  6.10 MBytes  5.11 Mbits/sec  766             sender
[  5]   0.00-10.00  sec  5.59 MBytes  4.69 Mbits/sec                  receiver
```

Figure 6.16: Connection test between Host 6 and 7(Internet Access)

The same happens if we define the client as host 1. As for any host outside the service that tries to access the Internet, access is denied.

6.3 Conclusion

This chapter describes the tests applied to the developed framework. Initially, it was detailed how to generate the test environment, with three switches and seven hosts (one host representing the Internet access). Once the environment was created, it was verified that there were no active services. Next, the user demonstrated a way to create a service using the voice assistant, Mycroft. Then, several ways of interacting with the different questions of the assistant were presented, and, in the end, it was proved that the system

could reach any state of the state diagram presented in section 4.4, finally reaching the success case. Once the service was created, we used, again, the iperf tool to prove its existence.

Chapter 7

Conclusion and Future Work

With the functionalities that virtual assistants have been offering, the horizons where they can be applied are increasing. This dissertation aims to implement features for a voice assistant in the context of industrial environments. Furthermore, the user experience improves through voice interfaces since he does not need to know the area.

The implementation of the project focused on developing a dialogue for the virtual assistant, Mycroft, that allows the user to orchestrate networks. From the dialogue, all the logic of the implementation of the application was made, from validation processes, conflicts and compilation, to the installation and later monitoring. As explained throughout the document, there was a need to dedicate more time to the design of the system (namely, the state machine and the intent structure) since a well-defined basis positively influenced the smooth running of the system. It was necessary to dedicate this time, which eventually exhausted the time allotted for other stages, which was the case of the time allotted for the implementation of the monitoring phase. This phase consisted of applying the concepts studied of ML in the reviewed literature to solve problems on the network without the need for human intervention. However, the research done will serve for future use.

Looking at the initially proposed objectives, with the help of Mycroft, a voice assistant based on voice recognition models, we identify the user's actions. Then, we mapped these intentions to understand the objective and implement it, and finally, we integrated the whole system to demonstrate that the functionalities are developed.

A test environment was created, and then the use case of service creation was applied. The different ways for the user to interact with the system to make the request were presented. Besides, all the possible causes of failure and success were presented. The presented results showed that the user could successfully control the network through voice commands, namely creating services. Advances in this technology bring benefits both to companies, who see the services Information technology (IT) become more relieved, and users can enjoy the service more quickly with the assistant's help.

The presented system can constantly be improved with more features for the user to orchestrate networks, and it can also be updated with new technologies that make the system more efficient. Improve the dialogues between the virtual assistant and the user to make a more user-friendly interface. Implement the monitoring phase, do more research on optimization algorithms and problem identification in networks, and then implement these algorithms based on the study done.

References

[1] What is the adapt intent parser? https://mycroft-ai.gitbook.io/docs/mycroft-technologies/adapt, last accessed on 12/2020.

[2] Pedro Amaral, Joao Dinis, Paulo Pinto, Luis Bernardo, Joao Tavares, and Henrique S. Mamede. Machine learning in software defined networks: Data collection and traffic classification. *Proceedings - International Conference on Network Protocols, ICNP*, 2016-December(February 2019):91–95, 2016.

[3] Isaac Ampratwum. An Intelligent Traffic Classification based optimized routing in SDN-IoT : A Machine Learning Approach, 2020.

[4] Tecnologia: Assistentes virtuais e o futuro da iot. https://transformacaodigital.com/tecnologia/assistentes-virtuais-e-o-futuro-da-iot/, last accessed on 12/2020.

[5] Assistentes virtuais: Como as empresas estao lidando com o avanco dessa tecnologia. https://digital.consumidormoderno.com.br/assistentes-virtuais-ed242/, last accessed on 12/2020.

[6] Abdelhadi Azzouni, Raouf Boutaba, and Guy Pujolle. NeuRoute: Predictive Dynamic Routing for Software-Defined Networks. *arXiv*, 2017.

[7] Douglas Comer and Adib Rastegarnia. OSDF: A framework for software defined network programming. *CCNC 2018 - 2018 15th IEEE Annual Consumer Communications and Networking Conference*, 2018-Janua(October 2017):1–4, 2018.

[8] Apache couchdb. https://couchdb.apache.org, last accessed on 12/2020.

[9] On the optimization of dijkstra's algorithm. https://arxiv.org/pdf/1212.6055.pdf.

[10] Experiential networked intelligence (eni); context-aware policy management gap analysis. https://www.etsi.org/deliver/etsi_gr/ENI/001_099/003/01.01.01_60/gr_ENI003v010101p.pdf.

[11] Zhong Fan and Ran Liu. Investigation of machine learning based network traffic classification. *Proceedings of the International Symposium on Wireless Communication Systems*, 2017-August:1–6, 2017.

[12] Yanjun Li, Xiaobo Li, and Osamu Yoshie. Traffic engineering framework with machine learning based meta-layer in software-defined networks. pages 121–125. Institute of Electrical and Electronics Engineers Inc., 12 2014.

[13] Cristian Cleder Machado, Juliano Araujo Wickboldt, Lisandro Zambenedetti Granville, and Alberto Schaeffer-Filho. Policy authoring for software-defined networking management. *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, pages 216–224, 2015.

[14] Abdehamid Abdelhadi Mansor, Wan M.N.Wan Kadir, Toni Anwar, and Shamsul Sahibuddin. Analysis of adaptive policy-based approach to avoid policy conflicts. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, 1:754–759, 2012.

[15] Abdehamid Abdelhadi Mansor, Wan M.N.Wan Kadir, and Hidayah Elias. Policy-based approach for dynamic architectural adaptation: A case study on location-based system. *2011 5th Malaysian Conference in Software Engineering, MySEC 2011*, pages 171–176, 2011.

[16] An overview of machine learning and its applications. `https://www.researchgate.net/publication/289980169_An_Overview_of_Machine_Learning_and_its_Applications`, last accessed on 12/2020.

[17] An introduction to ml/ai. `https://www.cisco.com/c/dam/m/cs_cz/training-events/webinars/tech-club-webinars/Artificial-Intelligence-Machine-Learning-Deep-Learning.pdf`, last accessed on 12/2020.

[18] Supervised vs unsupervised vs reinforcement. `https://www.aitude.com/supervised-vs-unsupervised-vs-reinforcement/`, last accessed on 12/2020.

[19] Moscow prioritization. `https://www.productplan.com/glossary/moscow-prioritization/`, last accessed on 01/2021.

[20] Mycroft. `https://mycroft.ai`, last accessed on 12/2020.

[21] Network intent composition. `https://github.com/opendaylight/nic`, last accessed on 12/2020.

[22] Controlador opendaylight. `https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2018_2/opendaylight/`, last accessed on 12/2020.

[23] Introduction: What is onos? `https://wiki.onosproject.org/display/ONOS/ONOS`, last accessed on 12/2020.

[24] Onos platform: An overview. `https://opennetworking.org/onos/`, last accessed on 12/2020.

[25] Intent framework. `https://wiki.onosproject.org/display/ONOS/Intent+Framework`, last accessed on 12/2020.

[26] Policy-based network management. `http://www.ittoday.info/Articles/Policy-Based_Network_Management/Policy-Based_Network_Management.htm`, last accessed on 12/2020.

[27] Meenaxi M. Raikar, S. M. Meena, Mohammed Moin Mulla, Nagashree S. Shetti, and Meghana Karanandi. Data Traffic Classification in Software Defined Networks (SDN) using supervised-learning. *Procedia Computer Science*, 171(2019):2750–2759, 2020.

[28] 94% of businesses will use iot by the end of 2021: Microsoft report. `https://theiotmagazine.com/94-of-businesses-will-use-iot-by-the-end-of-2021-microsoft-report-cf94ad11f173`.

[29] Mohammad Reza, Mohammad Javad, Seyed Raouf, and Reza Javidan. Network Traffic Classification using Machine Learning Techniques over Software Defined Networks. *International Journal of Advanced Computer Science and Applications*, 8(7), 2017.

[30] Mohammed Sameer and Bhargavi Goswami. Experimenting with ONOS scalability on software defined network. *Journal of Advanced Research in Dynamical and Control Systems*, 10(14):1820–1830, 2018.

[31] Wikipedia-software defined networking. `https://pt.wikipedia.org/wiki/Software_defined_networking`, last accessed on 12/2020.

[32] What-is-sdn. `https://www.ciena.com.br/insights/what-is/What-is-SDN_pt_BR.html`, last accessed on 12/2020.

[33] O que é sdn? entenda o conceito de software defined network. `https://stefanini.com/pt-br/trends/artigos/entenda-o-conceito-de-software-defined-network`, last accessed on 12/2020.

[34] O que sao redes definidas por software (sdn) e quais as vantagens? `https://blogbrasil.westcon.com/o-que-sao-redes-definidas-por-software-sdn-e-quais-as-vantagens`, last accessed on 12/2020.

[35] Sepia. `https://sepia-framework.github.io`, last accessed on 12/2020.

[36] Susi. `https://dev.susi.ai`, last accessed on 12/2020.

[37] Sebastian Troia, Alberto Rodriguez, De Dios, Rodolfo Alvizu, Francesco Musumeci, and Guido Maier. Machine-Learning-Assisted Routing in SDN-based Optical Networks.

[38] Vijay Varadharajan, Kallol Karmakar, Uday Tupakula, and Michael Hitchens. A policy-based security architecture for software-defined networks. *IEEE Transactions on Information Forensics and Security*, 14(4):897–912, 2019.

[39] Changhe Yu, Julong Lan, Zehua Guo, and Yuxiang Hu. DROM: Optimizing the Routing in Software-Defined Networks with Deep Reinforcement Learning. *IEEE Access*, 6:64533–64539, 2018.