Master Degree Course in Computer Engineering

Master Degree Thesis

A framework for automatic Network Security Functions selection and placement in NFV/Cloud context

Supervisors
prof. Riccardo Sisto
prof. Guido Marchetto
dott. Fulvio Valenza
dott. Jaloliddin Yusupov

Candidate
Giuseppe SISINNI
student ID: 241932

ACADEMIC YEAR 2018-2019

Summary

Nowadays, electronic device are increasingly widespread and used by all, young and old. They allow us to satisfy our desire to know, who has never done a search on the Web? They give us the opportunity to work remotely and, moreover, we use them for online shopping and to follow our team of the heart. But the most important thing is that they allow us to stay in contact with our loved ones that very often are miles from us. All these things are made possible by the fact that all devices are connected in the world through a large network: the **Internet**.

None of us care how this is possible but setting up the network to allow this is not easy at all. This is the task of the network administrators and their work is not so easy. The network is huge and consists of many and many devices mostly different and proprietary. Each device, therefore, requires specific knowledge to be configured. As a consequence of this, the network appears to be difficult to manage, monitor and slow to react to failures and security attacks. These motivations have led, in recent years, the development of two new paradigms: Network Function Virtualization (**NFV**) and Software-Defined Networking (**SDN**).

Network functions become virtual (**VNFs**): the software part, decoupled from the hardware, is allocated in virtual machines. The network becomes centralized: the control plane is separated from the data plane. In these ways the functions can be dynamically added and removed on-demand of customers needs. The whole network is under one control point. These are a great advantages because the operational cost, the response time and the administration tasks are reduced and the limits of the current network infrastructure are bypassed.

In each sub-network, managed by an administrator, there are a number of users, each with different requests and specific needs. Each user can request special care and the administrator has to satisfy them by treating them independently. He can do this writing a security **policy** for each of them. The security policy is a collection of policy **rules** where each rule is designed to handle a particular request. As an example, the administrator will write a policy dedicated to a father in order to forbid his child to go to social networks. But the work does not end here. In fact, after having defined the various policies, it is necessary to choose the network functions suitable for this purpose. This choice must be the best possible and this is not simple because every network function has costs: economic and physical resources used, such as ram, cpu and disk. Once selected, the functions have to be allocated between the available physical hosts and this is not an easy choice because it is necessary to ensure that the chosen hosts have the required resources available.

Nowadays an automatic mechanism does not exist to easily manage this situation and from this consideration comes the inspiration that has led to this thesis which proposes

a framework that can meet the work of network administrators, helping them to define policies and choose the functions to be used. Proposing an innovative way to address customer needs and to automatically configure the Service Chain.

The developed framework, *Verifuse* (**VERI**net **FU**nction **SE**lection and placement), can automatically choose how many and which security functions are needed, after having defined a set of policies written by one or more administrators for all users of the same network, and to allocate them among the physical servers available. Therefore, this thesis also proposes the following models made using the **XML language**: the **Policy Repository**, to allow administrators to express policies using the HPL language; the **Catalog of NSFs**, containing the list of all available network functions; **Capability**, key concept of the approach developed; **Hosts**, the physical servers available.

The idea is based on the concept of **Capability**: a set of features that share network functions and that enable certain policies to be met. Each network function supports one or more capabilities and each policy rule requires one. For example, Internet traffic control requires the capability Packet Filter.

The framework first allows administrators to define the policies and to list functions and hosts available, then analyzes the policies to derive the capabilities and finally selects and allocates the necessary network functions between physical hosts, optimizing the choice according to the parameters chosen by the user.

The selection and optimization phase is carried out using the ILP (Integer Linear Programming) solver Gurobi. Capabilities, functions, hosts and the relationships between them have been modelled using mathematical formulas and models that Gurobi tries to solve by looking for the best solution. The choice of the solution is guided by the parameters that the user chooses to optimize, for example the RAM consumption by functions.

The framework provides everything you need to define and analyze policies, allowing you to create your own catalog of functions and your physical host infrastructure. After choosing the optimization criterion, Verifuse automatically selects which network functions are needed to meet the policies. Therefore, it chooses functions, starting with policy analysis, and allocate them among the available hosts. The choice is optimized, it is the user who defines the parameters and the priorities. The modules scale very well and have been designed in such a way as to be independent and to be easily expandable in the future. In the future, in fact, we could improve the model of the hosts in such a way as to allow a better selection in the phase of optimization.

Acknowledgements

First of all, I want to acknowledge the supervisors of my thesis that are the professor Sisto and the professor Marchetto. I also want to thank Fulvio and Jalol who were extremely available every day and helped me a lot with their important advice.

I would like to thank all those people who believed in me during all these years. Especially to my family and in particular to my father and my mother who, with their countless sacrifices, allowed me to study so far from home, allowing this great moment in my life to become a reality. They support me every moment of my life, I will never stop thanking them.

Then, I thank my brother Daniele, my cousin Giuseppe, all my friends and all my colleagues who made these years easier by making the lessons more fun.

And last, but not the last, I would like to thank my love Roberta who with her love and patience has always been close to me, especially in the most difficult moments where everything seemed lost giving me the strength to go on.

I ended a phase of my life. They have been difficult years, leaving home so young is not easy especially if you find yourself in a city a thousand kilometers away from home with no family near. They have been years that have marked me and changed a lot. I will never forget them. It has been a path that I would do again, I would recommend it to everyone.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays electronic devices, such as computers and mobile phones, are used at any time of the day, both in our private life and in our working life. They allow us to satisfy our desire to know, who has never done a search on the Web? They give us the opportunity to work remotely and, moreover, we use them for online shopping and to follow our team of the heart. But the most important thing is that they allow us to stay in contact with our loved ones that very often are miles from us, in most cases we address them through a video call to feel them closer unlike the classic telephone call. All these things are made possible by the fact that all devices are connected in the world through a large network: the **Internet**.

Each of us uses the network to do these things easily, without worrying about how it happens and how it is possible all this. But setting up the network to allow this is not easy at all. Network administrators have a lot to do every day. This is because the network consists of countless devices that need to be added and configured according to user requests. The configuration and management of these devices is not simple and this is because the applications can be innumerable and very different from each other. There are also distinct types of devices and each requires specific knowledge to be configured correctly. The network is, therefore, difficult to manage, monitor and slow to react to failures and security attacks. These motivations have led, in recent years, the development of two new paradigms: *Network Function Virtualization* (**NFV**) and *Software-Defined Networking* (**SDN**).

The *NFV* aims to virtualize the network functions, which take the name of *Virtual Network Functions* (**VNFs**). The functions are allocated in virtual machines therefore they are separated from the hardware that becomes a simple host. In this way they can be dynamically added and removed on-demand of customers needs. This is a great advantage because the operational cost, the response time and the administration tasks are reduced.

The *SDN* has the aim to make the network directly programmable in order to improve network performance and monitoring, bypassing the limits of the current network infrastructure. To achieve this goal, it breaks the vertical integration and divides the control plane from the data plane. Therefore, the control logic (control plane) is centralized and is entrusted to a specific unit that takes the name of SDN Controller.

Each network administrator is responsible for setting up a given subnet where there are numerous users. Each user can request special care and it is the task of the administrator to satisfy them. Each user must therefore be treated independently. That is why the administrator must write a **security policy** for each of them.

The security policy is a collection of policy rules where each rule is designed to handle a particular request. As an example, how can the network be configured by an administrator if a father wants to forbid his child to go to social networks? The administrator will write a policy dedicated to the father. This policy will consist of a set of rules, including the one that prevents the child from misusing social networks.

But the work does not end here. In fact, after having defined the various policies, it is necessary to choose, among the available network functions, those suitable for this purpose. This choice must be the best possible and this is not simple because every network function has costs: economic and physical resources used, such as ram, cpu and disk. Once selected, the functions have to be allocated to the available physical hosts and this is not an easy choice because it is necessary to ensure that the chosen hosts have the required resources available.

Managing each customer individually is difficult and much more when the number of users increases. The context gets more complicated from an administrative point of view and can lead to the allocation of an improper number of necessary resources. To this end, nowadays an automatic mechanism does not exist to easily manage this situation and from this consideration comes the inspiration that has led to this thesis.

The aim of this thesis is precisely to propose a framework that can meet the work of network administrators, facilitating them in their task, proposing an innovative way to address customer needs and to automatically configure the Service Chain. In other word, to offer the *Verifuse* (**VERI**net **FU**nction **SE**lection and placement) framework that can automatically choose how many and which security functions are needed, after having defined a set of policies written by one or more administrators for all users of the same network, and to allocate them among the physical servers available. Therefore, this thesis also proposes the **models** of the policy repository, the catalogue of functions, capabilities and hosts.

The thesis is structured in the following chapters:

- **Chapter 2**, will describe the background in which this thesis has been developed. We will talk about virtualized networks (NFV, SDN, Service Chain), network security policies, Z3 and Gurobi tools;

- **Chapter 3**, the problem will be introduced and will be described in which chapters it has been approached;

- **Chapter 4**, will describe the approach used to implement the framework;

- **Chapter 5**, the models realized through the use of the XML language will be shown. Models make up the policy repository, function catalog, capabilities and hosts;
- **Chapter 6**, will describe the selection and optimization phase implemented in the SAP module;
- **Chapter 7**, will be shown the implementation, will be analyzed the packages and will be presented the REST API;
- **Chapter 8**, will be analyzed the results obtained after performing scalability tests;
- **Chapter 9**, the conclusions on the carried out job will be written.

Chapter 2

Background

This chapter will describe the background in which this thesis has been developed. In particular, the new paradigms of *NFV* and *SDN* will be analysed in detail because they are the two technologies that are having the power to change significantly the computer networks in recent years. They are two very different but perfectly complementary technologies, from their synergy comes the concept of *Service Chain*. The concept of policy will then be introduced and it will be discussed how it can be used in the area of security and access control. As last thing they will come introduced the two tools *Z3* and *Gurobi* that constitute the heart of the developed framework.

2.1 Software and Virtualized Network

2.1.1 Network Function Virtualization (NFV)

Nowadays computer networks continue to grow in size quickly and their management, by the various Internet Service Provider (ISP), is increasingly difficult and slow. For this reason a new paradigm that tries to solve these disadvantage was born: Network Function Virtualization is also known as NFV.

The traditional network is composed of numerous devices, each of which is dedicated to a specific function (e.g. firewall, NAT, proxy, VPN, etc.). For this reason the management as well as the configuration of the network, based on customers needs, are laborious and expensive. Indeed, whenever the demand for a new service occurs, it is necessary to redeploy the network or to add more physical and dedicated devices. This also implies to find power and space in which allocate the boxes.

In this scenario, the possibility of being able to virtualize network and devices becomes a great advantage. Network Function Virtualization exploits the benefits deriving from the virtualization changing the way in which the network is conceived. Defines a virtualized infrastructure for network functions so, in this way, is possible to reach the main objective: decoupling software from hardware.

These functions are named Virtual Network Functions (VNF) and are implemented as virtual machines. In this way they can be dynamically added and removed on-demand of customers needs. This is a great advantage because the operational cost, the response time and the administration tasks are reduced.

When the request for a new service occurs, is not necessary, adding a new hardware but just starting a new instance of a VNF. Therefore, is also possible testing an innovative service at lower risk because is not necessary to change the architecture of the network but only starting the virtual machine dedicated.

The NFV allows great advantages in various aspects:

- **Flexibility** Administrators can easily and quickly deploy or modify services. Providers can now choose between many vendors and have the flexibility to select the hardware capacities that are optimal for their network architecture and planning. They are not bound to choose a specific vendor and his proprietary hardware;

- **Cost** Maintain a network is very expensive, in this way is possible reduce management and hardware costs since all the functions are virtualized on virtual machines. NFV uses regular COTS (Commercial Off-the-Shelf component) hardware, network operators have the freedom to choose and build the hardware in the most efficient way to satisfy their needs and requirements. This allows ISPs to provide services to customers at a lower price;

- **Scalability** Until this moment, increasing the traditional network equipment's capacity has taken time, planning, and money. But, with this new approach, the network is now able to scale autonomously on-demand of needs. In fact, if any, of the VNFs requires additional CPU, storage, or bandwidth it can be requested from the VIM and allocated to the VNF. All this reduces also the time needed to deploy a service;

- **Security** This is one of the major challenges in networking and this new paradigm tries to help with the possibility to add easily new security function without having to buy a specific physical device.

It's possible to conceive that this new approach can be used to manage also Network Security Functions (NSF) but automating their support and configuration is fundamental. This is the aim of Thesis.

Framework

Inside the European Telecommunication Standards Institute (ETSI)[1], which is an independent standardization group, was formed the working group ETSI Industry Specification Group for Network Functions Virtualization (ETSI ISG NFV) which is in charge to develop requirements and architecture for virtualization of various functions within telecommunications networks.

[1]ETSI was created in 1988 by the European Conference of Postal and Telecommunications Administration (CEPT) and is officially recognized by the European Commission and the EFTA Secretariat. It is composed of more than 800 member organizations distributed in 65 counties and five continents. One of its main aims is to enable interoperability in a multi-vendor, multi-network, multi-service environment for this purpose its standards are designed for interoperability from the very beginning.

The ETSI ISG NFV helps, by setting terminology[1], requirements and architecture specifications for hardware and software infrastructure needed, to make sure virtualized functions are maintained how described in [2].

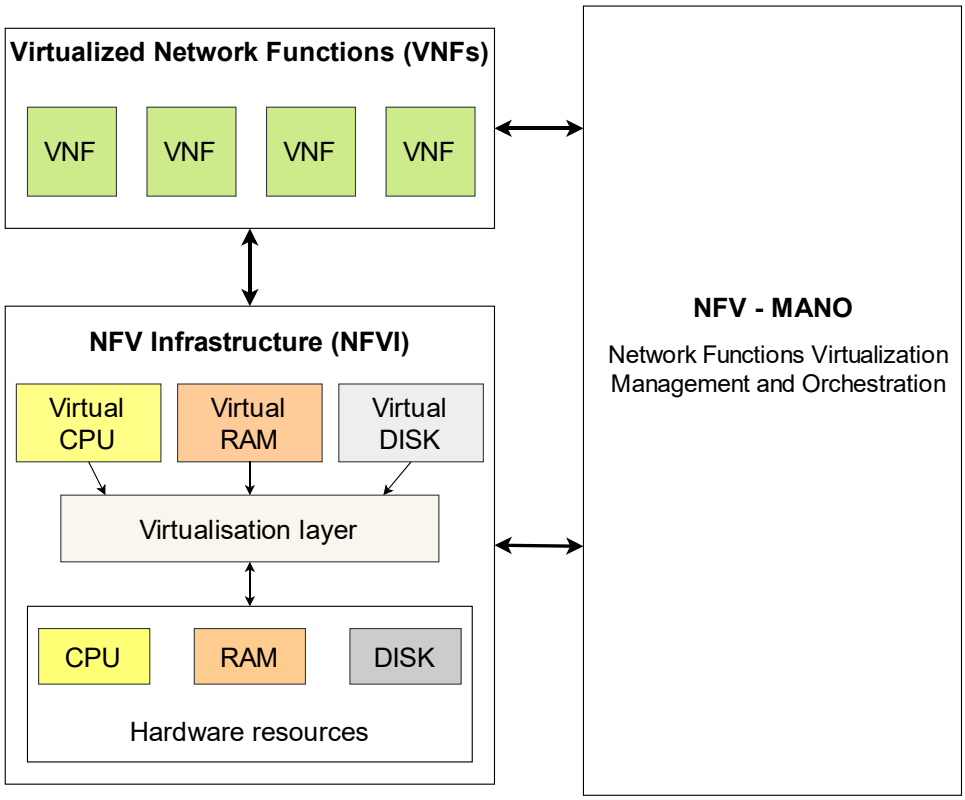

Figure 2.1: High-level NFV framework

As it is possible to see in the Fig. 2.1, the framework proposed by ETSI ISG NVF is composed at a high-level view of the following blocks:

- **Virtualised Network Functions (VNFs)** that are the software implementation of the various network functions that were physical devices until today. They are allocated in the NFVI;
- **NFV Infrastructure (NFVI)** that represent the infrastructure, and so the physical resources, in which the VNFs can be executed. In fact, the hardware resources, as compute, storage and network, are in common and must be shared between the VNFs to be instantiated. Above these physical resources there is a Virtualization Layer with the aim to virtualize the resources. In this way each VNF can access it in reserving way;
- **NFV Management and Orchestration (NFV MANO)** it is in charge of monitor, manage and orchestrate all elements inside the framework: computing, networking, storage and virtual machine (VM) resources. It also manages the life-cycle of VNFs.

In particular, as described in Fig. 2.2 it is composed by the following elements:

- **NFV Orchestrator (NFVO)** it is in charge of the resource orchestration in order to ensure that there are adequate compute, storage and network resources available to provide a network service. At this aim it can work with the VIM or directly with NFVI. In this case it can coordinate, authorize, release, and engage them without interacting with any specific VIM. It is responsible for loading new network services (NS) and managing its life-cycle as well as uploading VNF packages.
- **VNF Manager (VNFM)** it is responsible for the lifecycle management of VNFs. In particular, it is in charge to instantiate new VNFs, to scaling them, to update or upgrade them and to terminate them.
- **NFV Virtualized Infrastructure Manager (VIM)** it is responsible for managing the virtualized infrastructure. It holds traces of the physical resources devoted to those virtual by the NFVI so it is able to orchestrate the allocation, the updating, the release and the recovery of the resources in a way to optimize their use. It manages security group policies to ensure access control. It also deals with collecting information on performance and failures through notifications and managing catalogs of hardware and software resources.

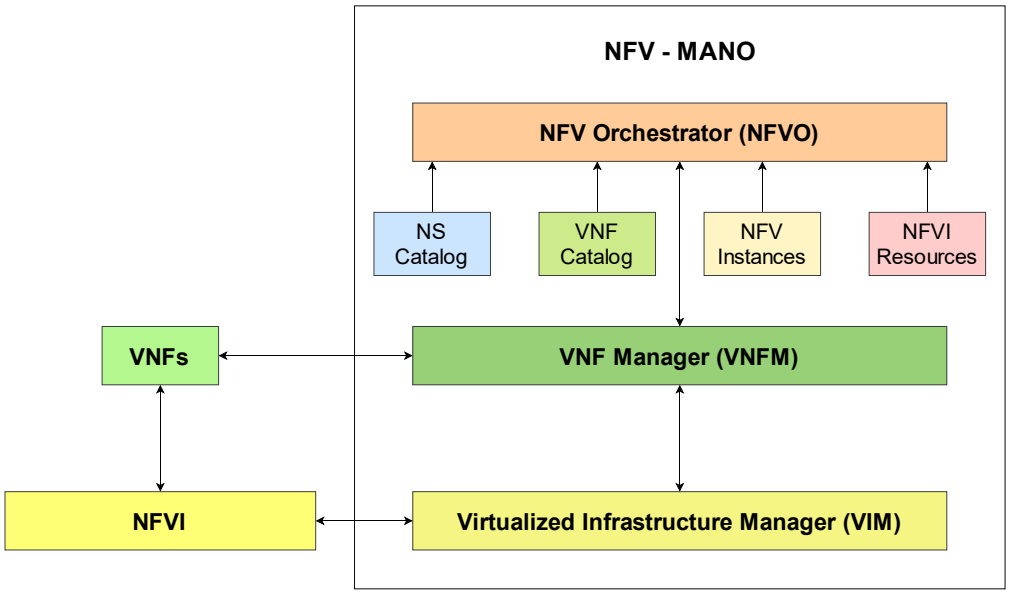

Figure 2.2: NFV MANO

2.1.2 Software-Defined Networking (SDN)

The birth of first computers, which was followed by their rapid spread in universities and research labs, soon led to the need to be able to share documents in the same room for research purposes. For this reason, in the late 1960s was born Internet in the form of LAN(Local Area Network).

For the first time computers were connected together and, since this happened in a limited area, there were not many problems related to the safety or to the quality of the service. In fact, many protocols were created without taking into consideration these problems, which have become big and open issues today.

From that moment, the Internet has spread more and more over the course of time becoming a worldwide phenomenon that has forever changed the history of man.

All this has led to the creation of a digital society where anything is connected and accessible from everywhere in an easy way by all the people; in fact no special technical knowledge is required to use the various network services. Just think of the fact that nowadays also the children are able to use the Internet without critical problems.

But for the network administrators this is not so easy. In fact, even if in the original idea Internet was conceived to be simple as possible, in reality the traditional IP networks are complex and very difficult to manage. This because the Internet network is composed of packets that travel in the world through network devices connected between them by complex rules.

Network operators must configure individual network device separately, using low-level and often vendor-specific commands, making difficult configure the network, according to predefined policies and to reconfigure it to respond to faults. Indeed, the current network is not in charge to respond automatically and to configure itself autonomously.

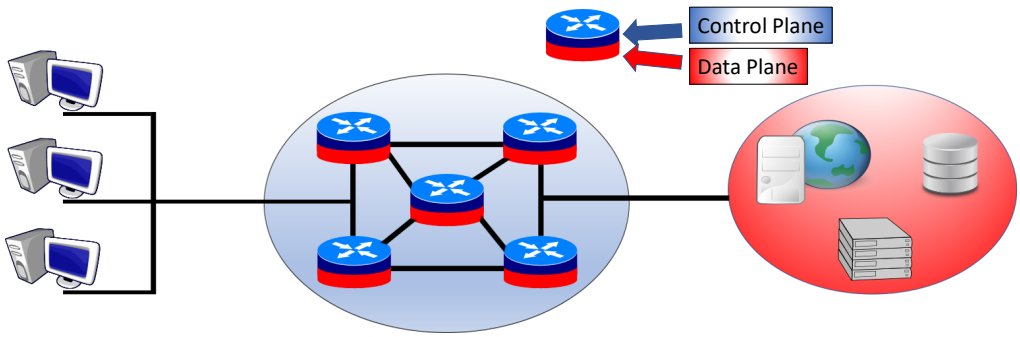

Figure 2.3: Traditional Network

In the traditional network, Fig. 2.3, the movement of packets, and so the traffic plane, is managed by each single network device that therefore performs three distinct activities:

- **Control plane** is the part of a network that carries signaling traffic. The knowledge of the topology of the network, obtained by exchanging information through protocols, allows it to be responsible for the routing and so to route packets by establishing connections between routers in the network;

- **Data plane** also known as forwarding plane, it is in charge of actually forwarding packets according to the decision made from the control plane;
- **Management plane** is considered a subset of the control plane because carries administrative traffic with the aim to provide management, monitoring and configuration services.

The traditional network is, therefore, vertically integrated. That means that the control plane and the data plane are wrapped together, in the same device, and this reduces the flexibility of the network. In this context was born the idea that led to a new way to see the network: Software-Defined Networking (**SDN**).

The SDN has the aim to make the network directly programmable in order to improve network performance and monitoring, bypassing the limits of the current network infrastructure. To achieve this goal, it breaks the vertical integration and divides the control plane from the data plane. Therefore, the control logic (control plane) is centralized and is entrusted to a specific unit that takes the name of SDN Controller.

The SDN Controller communicates with the devices distributed on the network that, from network switches, become simple forwarding devices with the only aim to forward packets based on some rules defined by the controller. As shown in Fig. 2.4.

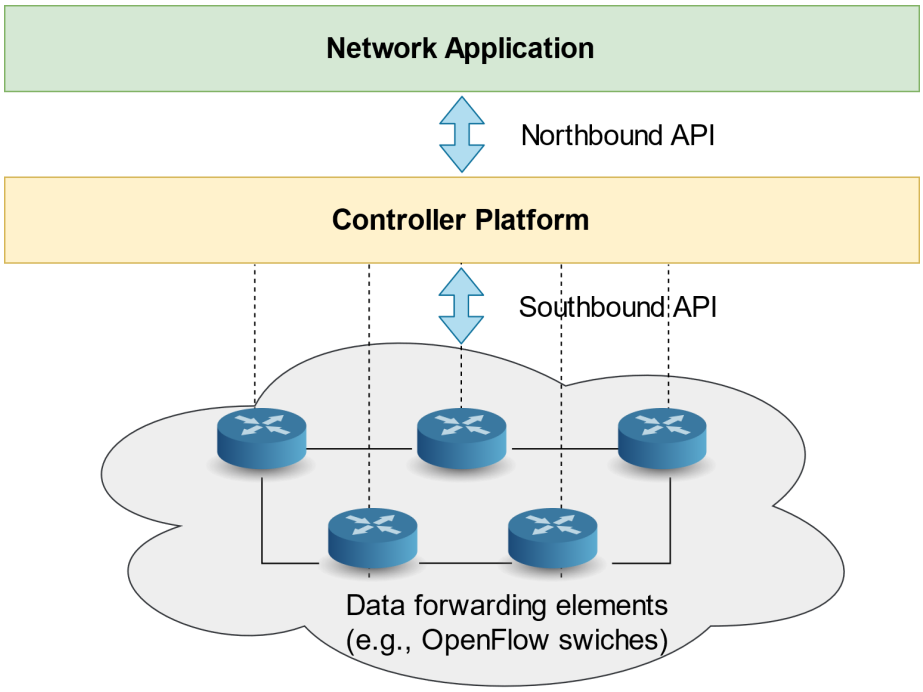

Figure 2.4: SDN architecture

In this way the policy enforcement, the network configuration and its evolution are simplified.

In fact, the configuration of the whole network can be carried out, in a programmatic way, in an only node that is the controller. It is important to emphasize that a logically centralized model does not postulate a physically centralized system. Indeed, if it was so,

would be problems regarding performance, scalability and reliability of the network. For these reasons the control plane should be physically distributed.

As a direct consequence of the fact that there is an only centralized logical node, the whole network is capable to respond automatically to the traffic needs and to reconfigure itself in case of faults. But if all this leads to significant advantages, on the other hand, there are some disadvantages regarding scalability and, particularly, security: there is a single point of failure. This poses new issues and opens up new controversies on this new paradigm, which is revolutionizing networks and opening up new possibilities. As a simple example, just think of the fundamental contribution to the development of 5G.
As can be seen better in Fig. 2.5, the SDN architecture is composed of three layers:

- **Application Layer** can be considered the brains of the network. Includes the applications, programs, that via northbound application programming interfaces (APIs) require the necessary resources to use by the SDN controller;
- **Control Layer** contains the SDN controller, so this layer receives requirements from the SDN Application layer and forwards them to the networking components through southbound API, the most adopted is OpenFlow[2];
- **Infrastructure Layer** is composed of a set of networking equipment (switches, routers, etc.) with the only aim to forward packets, without the competence to take autonomous decisions.

[2]The OpenFlow, developed by the Open Networking Foundation (ONF), is a programmable network protocol for SDN environment, which is used for communication between switches and controllers. OpenFlow supports switches of different vendors that are called OpenFlow switches. The switch is composed by flow and group tables, which exploits to search and forward packets. It is also made up of OpenFlow channels that link it to an external controller. Using the OpenFlow switch protocol, the controller can modify, add, update, and delete flow entries in flow tables [3].

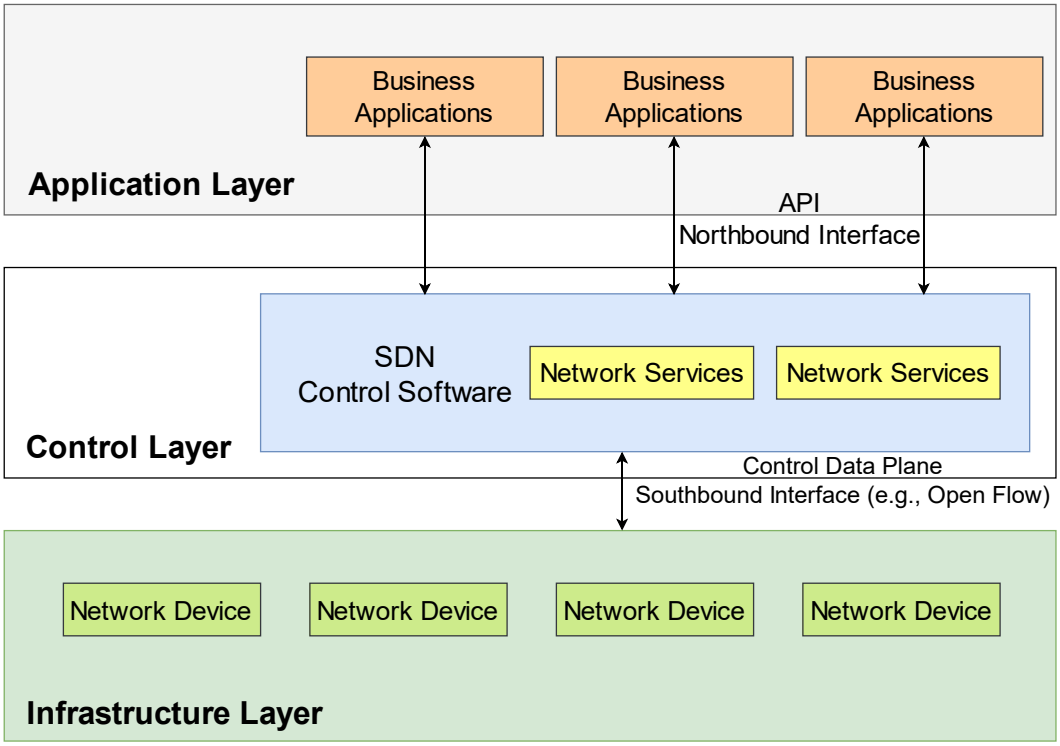

Figure 2.5: SDN architecture, layer view

2.1.3 Service Function Chaining (SFC)

As seen in the previous section, Software-Defined Networking and Network Function Virtualization are two new paradigms that are revolutionizing, in a completely different way, the network idea. Both SDN and NFV aim to develop a software-based approach to networking for more scalable, agile[3], and innovative networks.

These paradigms are mutually useful, but are independent of one another, or better they are perfectly complementary. For this reason, the idea of combining synergies within the network was born. The power of their union is seen more and is fully exploited with the concept of Service Function Chaining (SFC).

Whenever a demand for a new service occurs by the customers, operators must choose the best plan to satisfy them. This requires a lot of time and costs, but the advent of these new paradigms (Agile, SDN and NFV) brought significant advantages.

The Network Service offered to end users is composed by a various number of Service Function, which is a function that is responsible to manage the incoming packets and can act at different level and layers of a protocol stack. It is important that these Service Functions are ordered in a specific and logical way, because there are some constraints to be satisfied. For example, as is possible to see in Fig. 2.6, in this scenario it is important that the DPI is applied after the Firewall. The ordered set, of Service Function to be satisfied, takes the name of Service Function Chains[5].

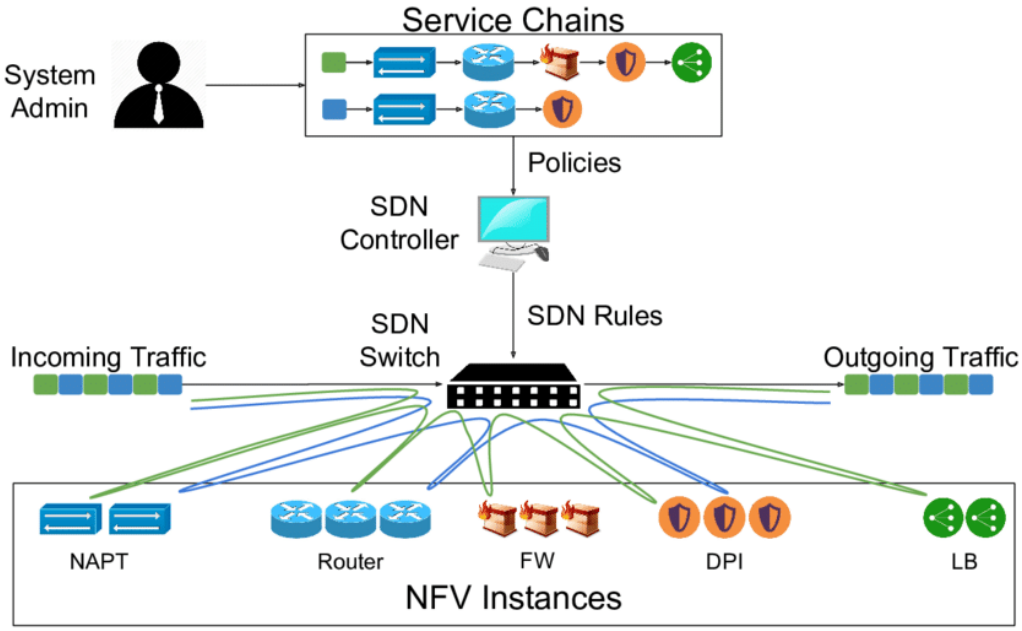

Figure 2.6: Service Function Chains

In this context is possible to see how the Software-Defined Networking and the Network Function Virtualization work perfectly together and are fully complementary.

[3] Agile is a software development strategy that involves the customer from the beginning and in each of its development phases, in order to obtain a high response to its requests and a rapid development of the final product [4].

Indeed, the administrator defines for each user, based on his needs, the Network Service, which is expressed as a chain of service functions. The service chain represents the application level in the SDN context, so this request is handled by the SDN controller. The SDN controller forwards to the lower infrastructure the correct commands to satisfy the chain and which network functions must be enabled. The infrastructure is composed of SDN switches containing the instances of the network function to be enabled to meet the requirements.

Thanks to this new approach the network results more dynamic and easy to manage. Also customer needs result easier to meet.

In this new complex scenario, is possible to set up and choose which NFV are necessary on any policy rule expressed by end users. As an example, how can the Service Function Chain be configured if a father wants to forbid his child to go to social networks?

The many problems, related to this scenario, will be addressed in details during the drafting of the thesis. In particular, they will be discussed in depth with importance in Chapter 4 while they will be introduced summarily in the next section.

2.2 Policy languages and Specifications

This section begins with the final question of the previous one; is it possible to configure the network in order to allow a father to monitor his child by prohibiting him access to social networks, in such that to do not distract himself during the study in the afternoon? In other word, is possible for the network administrator configure the network, or better the Service Chains, in order to satisfy all the additional requests made by customers?

Therefore, the network administrator is forced to accommodate many additional requests coming from the various customers. In fact, the administrator not only has to configure the network in order to guarantee the access to the Internet to the own customers and therefore that they are able to use the various services but he must also take into account further demands, and therefore constraints, which change the basic behaviour of the services offered. This is because now the administrator will not simply have to guarantee his client are able to access to the Internet, but will have to bind him to particular conditions (for example block specifics sites).

Managing all these possible situations is not so easy for the network administrator as each client may have different needs that need to be taken into account and met.

Moreover, the evolution of technology has led to the creation of increasingly complex systems and the management of these tools has also evolved. This last motivation has given a strong push towards the introduction of abstract models that allow the description and the management of the systems without going into details that are too specific for the administrator

The necessity of abstraction so that these situations are better managed by the system administrator has led to the definition of the concept of **Policy**.

2.2.1 Definition

The concept of **Policy** is, therefore, abstract and depends on the context of use; in this thesis we will deal with this concept in relation to computer network systems.

To define in precise way this concept is not much simple, in literature there are several definitions, the more important are the following:

- The current definition is given by the IETF (Internet Engineering Task Force) in the **RFC-3198** [6]:

  "Policy can be defined from two perspectives: (i) A definite goal, course or method of action to guide and determine present and future decisions. "Policies" are implemented or executed within a particular context (such as policies defined within a business unit); (ii) Policies as a set of rules to administer, manage, and control access to network resources [7]"

- Another definition is given in [8] by D.Clark e D.Wikson in the article *"A Comparison of Commercial and Military Computer Security Policies"*:

"Any discussion of mechanisms to enforce computer security must involve a particular security policy that specifies the security goals the system must meet and the threats it must resist. For example, the high-level security goals most often specified are that the system should prevent unauthorized disclosure or theft of information, should prevent unauthorized modification of information, and should prevent denial of service";

- Another definition is given in [9] by D. C. Robinson e M. S. Sloman in the article *"Domains: a new approach to distributed system management"*:

  "A management policy defines the set of rules to achieve certain objectives. For example, an access control policy'and a set of rules that define the resources that a user can access and a fault management policy defines where a fault should be reported and some recovery actions. The policy is defined by the system administrator".

The definition that best fits the purpose of this thesis is the first. For this reason, during the development of this thesis, when we talk about **Policy** we will refer to it as a set of **Policy Rules**, written by administrators, with the aim of monitoring and managing user requests.

This definition was in fact presented for the first time in the **RFC3060**[7], *"Policy Core Information Model"* (PCIM), which represents the object-oriented information model for representing policy information.

The **PCIM** defines two hierarchies of object classes:

- structural classes representing policy information and control of policies;
- association classes that indicate how instances of the structural classes are related to each other.

As described in the RFC3060, a network, which implements the policies, can be modelled as a state machine that uses policies to control, at all times, what is the status of the devices. The key points in the definition of the policy are as follows:

- Each *policy* is composed by a set of policy rules enabling its application;
- Each *policy rule* consists of a set of conditions and a set of actions;
- Multiple rules can be aggregated into *policy groups*;
- Multiple groups may be nested in a hierarchic way.

Let's see their meaning.

Conditions set

The set of conditions is constituted from that set of conditions that, if triggered, allow the execution of the rule. Therefore they specify the moment of application of the rule itself.

If they are assessed positively, the corresponding set of actions will be executed.

Actions set

The set of actions is constituted from that set of actions that are executed by the rule. It is possible to define a specific order in which actions should be executed or declare that the order does not affect execution

Hierarchic

The rules defined in the set can be characterized by a priority which allows to declare, therefore, a general main rule followed by more specific underlying rules. In this way you get a hierarchy of rules. This allows you to execute only the actions of the rule with greater priority when more rules are met.

Policy groups

Policies can be used individually or grouped together in groups called policy groups. These allow to represent the interactions between objects and their dependencies.

Categories

Policy groups and policy rules are divided by category according to their purpose: configuration policies, installation policies or security policies.

The aim of this thesis has been to place particular attention on those related to security in such a way as to allow or deny access to a specific resource by an applicant. Rules, conditions, actions and related data are kept in a logical container called *Policy Repository*.

2.2.2 Policy framework architecture

In the **RFC-2753** [10] is shown how a policy framework can be implemented using the following components:

- **Policy Repository**, this is the logical container. It is composed by rules, conditions, actions and related data. It contains all the policies of the system;
- **Policy Decision Point** (PDP), it is a logical entity that makes policy decisions for other network elements that require these decisions. It is in charge of establish whether or not a person has a permit, if it is legal or false, if it is still valid or revoked and so on. It is the point where policy decisions are made;
- **Policy Enforcement Point** (PEP), it is a logical entity that applies policy decisions. Therefore, it is a single point of access where the policy decisions are actually enforced. It defends the resource;
- **Policy Information Point** (PIP), it is a logical entity that provides the requested access information. This additional information depends on the context;
- **Policy Access Point** (PAP), it is responsible for providing the policy applicable to the requested access;

- **Subject**, he is the one who wants to access a particular resource. It is necessary to verify if there is a policy directed to him that manages his access to the resource;
- **Resource**, it is the object the user wants to access.

The workflow, visible in Fig. 2.7, consists of the following steps:

1. A *subject* applies to *PEP* for access to a given resource;
2. *PEP* to decide whether to allow access or not, communicate with the *PDP*;
3. *PDP* asks *PIP* for further information about the context;
4. *PDP* asks *PAP* if there are any policies that regulate the access of the subject on the resource in question. It is important to note that the *PAP* manages the policy repository, containing all policies. These last ones have been inserted in a time "0", that is to the birth of the system;
5. *PEP* receives from *PDP* the outcome of the decision;
6. *PEP* shall communicate the decision to the *subject*.

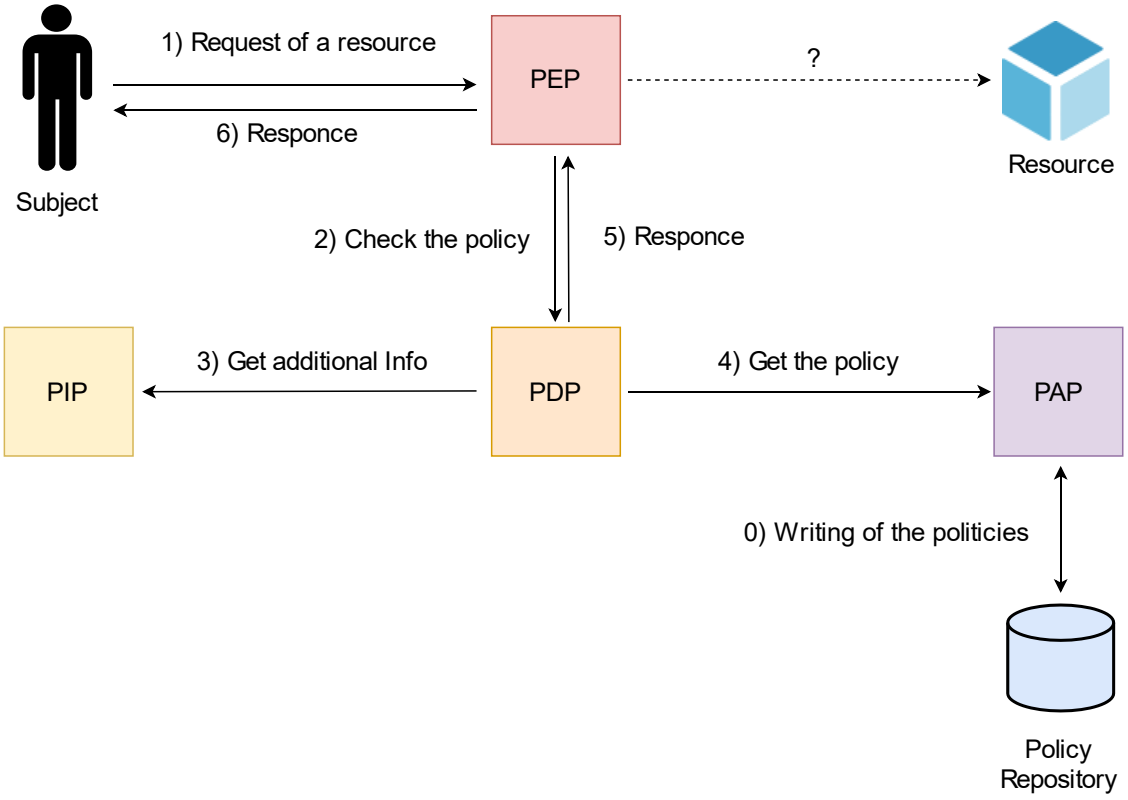

Figure 2.7: Policy framework architecture

This implementation gives a high-level view of the policy concept and how it can be used in practice. It helps us understand how important this concept is and how this mechanism can be used to enable or not an user to access a particular resource.

The work carried out by *Firmato* [11] and *Policy Manager* [12] has its roots precisely in this.

2.2.3 Policy languages

Now that it is clear how this mechanism uses policies as a key element in security and access control, the question is: how are these policies expressed? What kind of language is needed? Should the language used be closer to users (and in particular to the administrators) or to machines?

The answer is not as obvious as you might think. This is because if the language used is very close to the machine, then at a low level, it is difficult to understand for those who are not experienced. Also, on the market there are countless devices and each with its own proprietary language (e.g. Cisco) and this further complicates the situation.

On the other hand, if the language used is closer to the humans, surely it is understandable also for those who have no experience but it can be difficult, and not always possible, to adapt it for various devices.

One of the aims of this thesis has been precisely to deepen the study of the various languages in such a way as to model the policy repository and the catalogue of network functions. In such a way as to allow the use of these devices to all users, from those without experience to those who use these devices every day.

The languages used are divided into:

- **High-level Policy Language**(HPL), is an authorization language with the aim of expressing the user requirements by hiding the configuration details during the definition of the policies. It has been idealized as an user-oriented policy language that exploits sentences close to the natural language.

  For instance: "User *A* cannot access the Internet on Monday afternoon";

- **Medium-level Policy Language**(MPL), is an authorization language addressed to end users with good technical knowledge. It is an abstract language that is still vendor-independent, but allows expressing specific configurations in a generic format. Therefore, these configurations must then be translated into the proprietary language of the device to be configured;

- **Low-level Policy Language**(LPL), it represents the proprietary language that must be manually configured in the devices.

These languages will be discussed and studied with particular attention in the Sec. 4.2.

This problem, that is the choice of the language to be used so that policies can be expressed clearly and simply, has taken importance in recent times and many are the works

carried out in literature that guide them. One of these is definitely the European project **SECURED** which is the starting point of this thesis. It uses these languages in the field of security so much that it manly talks about *High-level Security Policy Language* (HSPL) and *Medium-level Security Policy Language* (MSPL). Security rules can be expressed either through the use of a high level language (the HSPL, easy for everyone) or through a medium level language (the MSPL, suitable only for experienced people in the field). In both cases the rules are expressed independently of the device and must be adapted. This introduces a level of abstraction that allows network administrators to focus only on policy rules.

A bit different is the concept of language implemented by **Firmato** [11] where a medium-level language is used. It tries to abstract the guidelines for firewalls making them regardless of the seller. These rules are then translated to a low level by a compiler that is responsible for generating configuration files based on the device. This mechanism introduces a new level of abstraction that leads network administrators to focus exclusively on firewall rules regardless of network topology.

2.3 Tools

2.3.1 Z3

Z3 is a tool developed by *Microsoft Research* with the aim of solving *Satisfiability Modulo Theories* (**SMT**) problems [13].

The SMT problem is a decision problem[4] for logical formulas expressed in first-order logic. These formulas are composed of sentences which are logically combined exploiting quantified variables. Each sentence consists in a predicate and a subject, which is represented by a variable.

As an example, if we have the following predicates:

$$\texttt{P} \triangleq \texttt{Socrate is a man}$$
$$\texttt{Q} \triangleq \texttt{is mortal}$$
$$\texttt{R} \triangleq \texttt{Socrate is mortal}$$

If we want generalize in order to obtain

$$\texttt{S} \triangleq \texttt{every man is mortal}$$

We can declare the x variable obtaining:

$$\texttt{P} \triangleq \texttt{is a man} \Rightarrow man(x)$$
$$\texttt{S} \triangleq \texttt{is mortal} \Rightarrow mortal(x)$$
$$\texttt{Q} \triangleq \texttt{every man is mortal} \Rightarrow \forall x(man(x) \rightarrow mortal(x))$$

Z3 is able to solve much more complicated formulas, consisting of variables (numeric or Boolean) and logical or mathematical expressions.

As an instance, if we define the integer variables x and y, we can write the following formula:

$$x + y < 10 \tag{3.3}$$

Z3 is able to determine if this equation is *Satisfiable* (SAT), which means if there are values to assign to the variables x and y in order to satisfy the inequality and in this case is also able to provide a use case (as an instance: x=5, y=3) or if it is *Unsatisfiable* (UNSAT) and which therefore cannot be satisfied.

Z3 allows also to impose some constraints which can be hard or soft.

- **Hard-Constraint**: they must be met obligatorily and therefore in case they cannot be satisfied the whole expression turns out to be UNSAT;

[4] A problem is decisional if it can be posed as a yes-no question of the input values.

- **Soft-Constraint**: Z3 tries to fulfill them after having satisfied those hard, in case it is not possible the expression is still SAT. This constraint is used to optimize the result.

Taking into consideration the formula (3.3) we can add the Hard-Constraint:

$$x \neq 5 \tag{3.3.1}$$

In order to oblige Z3 to do not use this value to solve the formula.

Therefore, we can define variables which take into account policies, capabilities, functions, servers and define predicate for their relationships. At this point we can define hard constraints in order to impose certain conditions (as an example: so that an NSF to be chosen it is necessary that it supports at least one required capability) and soft constraints in order to optimize the choice of functions (as an instance: to reduce the cost) and leave to z3 the task of finding a solution that is satisfiable.

2.3.2 Gurobi

Gurobi is a commercial optimization solver for *linear programming* (LP), *quadratic programming* (QP), *quadratically constrained programming* (QCP), *mixed integer linear programming* (MILP), *mixed-integer quadratic programming* (MIQP), and *mixed-integer quadratically constrained programming* (MIQCP).

It was developed in 2008 from a group of persons whose initials constitute its name: Zonghao **Gu**, Edward **Ro**thberg and Robert **Bi**xby.

It is the fastest and most powerful mathematical programming solver, available for all operating systems and programming languages. It enables users to state their toughest business problems as mathematical models and then finds the best solution out of trillions of possibilities.

As mentioned above, Gurobi is able to solve the following mathematical problems:

- **LP**, solves the mathematical model whose requirements are represented by linear relationships. Therefore, linear programming allows the optimization of a linear objective function, subject to linear parity and constraints of linear inequality. Its practicable region is a convex polytope and its objective function is a related function of real value defined on this polyhedron. A linear programming algorithm finds the point, in the polyhedron, where this function has the smallest (or greatest) value.

  The canonical form of a linear program model is the following:

$$\texttt{Maximize } c^T x$$
$$\texttt{subject to } Ax \leq b \texttt{ and } x \geq 0$$

  Where:

  – **Maximize/Minimize** is the aim;

  - **x** represents the vector to be determined;
  - **c** and **b** are vectors of known coefficients;
  - **A** is a matrix of known coefficients;

- **QP**, solves the mathematical model where the objective is to maximize/minimize a quadratic function of several variables subject to linear constraints on these variables.

  The canonical form of a quadratic program model is the following:

$$\begin{gathered}\texttt{Maximize } \tfrac{1}{2}x^TQx + x^Tx \\ \texttt{subject to } Ax \leq b\end{gathered}$$

  Where:

  - **Maximize/Minimize** is the aim;
  - **x** represents the vector to be determined;
  - **Q** is a real symmetric matrix;
  - **c** and **b** are vectors of known coefficients;
  - **A** is a matrix of known coefficients;

- **QCP**, solves the mathematical model where the objective is to maximize/minimize a quadratic function of several variables subject to quadratic constraints on these variables.

  The canonical form of a quadratic program model is the following:

$$\begin{gathered}\texttt{Maximize } \tfrac{1}{2}x^TP_0x + q_0^Tx \\ \texttt{subject to } \tfrac{1}{2}x^TP_ix + q_i^Tx + r_i \leq 0 \texttt{ for } i = 1, ..., m \\ \texttt{and } Ax = b\end{gathered}$$

  Where:

  - **Maximize/Minimize** is the aim;
  - **x** represents the vector to be determined;
  - P_i are matrix;
  - P_i and **b** are vectors of known coefficients;
  - **A** is a matrix of known coefficients;

- **MILP**, solves the mathematical problem similar to *LP* but the variables are not discrete;

- **MIQP**, solves the mathematical problem similar to *QP* but the variables are not discrete;

- **MIQCP**, solves the mathematical problem similar to *QCP* but the variables are not discrete.

Gurobi is used within the framework to solve a particular case of the ILP problem, the variables used are **binary** and only allow values 0 and 1.

Gurobi assigns unknown values by optimizing an objective function, can maximize or minimize it. The values assigned may be subject to constraints, for example, it can be imposed that they must be strictly positive.

Gurobi, unlike Z3, is optimized to solve multi-objectives. You can specify multiple objectives and decide whether to meet them hierarchically, blending them or mixing the two approaches. The difference is as follows:

- **Blended Objectives**, this approach creates a single objective by taking a linear combination of your objectives. This is possible providing a weight for each objective.

  As an example, if the model has the two objectives:

$$5 + 2x + 3y \tag{2.3.2.1}$$

$$x - y + 8 \tag{2.3.2.2}$$

  If weight -1 is assigned to equation 2.3.2.1 and weight 2 to equation 2.3.2.2, the following equation resulting to be optimized will be obtained:

$$\begin{aligned}(-1) * (5 + 2x + 3y) + 2 * (x - y + 8) &= -5 - 2x - 3y + 2x - 2y + 16 \\ &= 11 - 5y\end{aligned} \tag{2.3.2.3}$$

- **Hierarchical Objectives**, the hierarchical or lexicographic approach assigns a priority to each objective, and optimizes for the objectives in decreasing priority order. At each step, it finds the best solution for the current objective, but only from among those that would not degrade the solution quality for higher-priority objectives.

  You can also specify a degree of degradation. That is, it is possible to choose as the final solution, the solution obtained by optimizing the subsequent objectives, which has degraded the primary objective by a chosen value;

- **Combining Blended and Hierarchical Objectives**, it is also possible to set both a weight and a priority for each objective. This allows you to combine the blended and hierarchical approaches.

Chapter 3

Problem Statement

The aim of this thesis is to propose a framework that can meet the work of network administrators, facilitating them in their task, proposing an innovative way to address customer needs and to automatically choose which network functions are needed and where to allocate them. Allowing the administrators to define the policy repository, to create the catalog of functions available in the system and specify the physical hosts where virtual functions can be allocated.

The purpose was, therefore, to develop the *Verifuse* (**VERI**net **FU**nction **SE**lection and placement) framework that can automatically choose how many and which functions are needed, after having defined a set of policies written by one or more administrators, for all users of the same network, and to allocate them among the physical servers available.

All this has been possible through a long study of policies and previous works in literature. This study has led me to a new way of seeing policies and a revolutionary new way of associating policies with security functions. How this has been made possible and how it works will be better described in the **Chapter 4** but the basic idea is in the concept of **Capability**. By analyzing the various network functions I realized that they can be divided into groups because they share common characteristics. For example, firewalls are nothing more than packet filters. All network functions that have the characteristic of working at *level 4* and *filtering packets* then have the *Capability* to be *Packet_Filter_L4*. Each capability is therefore characterized by a series of features that make it unique, it represents all those functions that have a certain characteristic. Each network function can support multiple capabilities, one necessarily.

The capabilities allow, therefore, a connection with the network functions but what about the policies? The network administrator must write a **security policy** for each user who belongs to the network in such a way as to satisfy its requests. The security policy is composed by a collection of **policy rules** where each rule is designed to handle a particular request. Each rule is represented by a condition and an action that has to be performed (for example, block internet traffic) that in fact connects to a specific capability (as an instance, packet filter) and then to the use of a function (e.g., Iptable).

Therefore, starting from the policies the necessary capabilities are identified and from these they come to select the necessary functions. But how does this choice occur? According to which constraints the functions are chosen? Once chosen, where are they allocated? Is it possible to optimize these choices? All these questions will be answered in depth in the **Chapter 6**. What we need to know is that all this has been made possible by the use of the **Gurobi Integer Linear Programming (ILP) Solver**. Gurobi associates binary variables, with value 1 (in case of choice) and 0 otherwise, to the various elements (Network Functions, Capability and Host) and relationships between them. It assigns to all there variables some values which depend on some constraints expressed by mathematical formulas, systems of linear equations, that Gurobi tries to solve.

All these elements (Policy Repository, Catalog of Network Functions, Capabilities and Hosts), which will be presented in detail in the **Chapter 5**, have been modelled through the **eXtensible Markup Language (XML)** language. This is a markup language that allows to model in a very precise way, thanks to the use of constraints, all necessary data structures. For the realization of these models, of great importance was the use of the High-level Policy Language (HPL). Its use has been analyzed in the European project *SECURED* and improved. It has been expanded to allow detailed policy modelling. Now the network administrator can, with simplicity, write various and precise policies. Support any kind of situation that might occur.

As described in the **Chapter 7**, the framework has been implemented by exploiting the Java language, in particular the platform JAVA JDK 8 SE with the external tools Ant, Tomcat, Neo4j and Gurobi. REST APIs have also been developed to offer the potential of the framework to the outside. The framework has been designed to be fast, scalable and modular. For these reasons it has been divided into two key modules:

1. **CAID** (CApability IDentifier) which has the aim to analyze the policies, expressed by HPL, to identify which capabilities are needed to satisfy them.

2. **SAP** (Selection And Placement) which has the purpose to determine and select how which network functions are required to meet the required capabilities. It has also the task to allocate the instances of the functions between the available physical host.

The two modules have been developed to be completely independent, they do not communicate directly because the result produced by *CAID* turns out to be the input of a module external to the framework. The external module sends new data to the *SAP* module. The *SAP* module selects the necessary functions and allocates them among the available hosts. The presence of another module outside the framework justified the choice of the two modules in such a way as to ensure maximum modulation and interoperability.

The framework has been created to be practical and fast without penalizing the goodness of the solution. The synergy between the modules has led to excellent results that will be shown in the **Chapter 8**. Scaleability tests have been performed by varying the number of: policies, capability instances, functions and hosts. The number of inputs has been progressively increased up to one million. The large quantity of entries, resolved in

reasonable time, demonstrated the goodness of the solution and and the execution speed of the framework.

The thesis work has achieved the goal of developing the **Verifuse** framework. It provides everything you need to define and analyse policies, allowing you to create your own catalog of functions and your physical host infrastructure. After choosing the optimization criterion, Verifuse automatically selects which network functions are needed to meet the policies and allocate them among the available hosts. The choice is optimized, it is the user who defines the parameters and the priorities. The final considerations and possible future developments will be shown in the last **Chapter 9**.

Chapter 4

Approach

In this chapter, the approach taken to solve the problems mentioned in the previous chapter will be addressed. In particular, the adopted model, which concurs to formalize the security policies in order to be able to choose the network functions to use, will be described in the detail.

It will be discussed both how the *High-level Policies Language (HLP)* can be used to express the safety requirements of non-technical users and how the *Medium-level Policies Language (MLP)*, on the other hand, allows experienced users in the area to express their configurations.

Particular attention will then be given to the innovative approach allowing to complete the refinement process, which determines which network functions must be enabled, among those available, according to the policies required and where to allocate them in the physical servers. At the end of the chapter, it will be discussed the strategy used to optimize these choices.

4.1 Workflow

The main problem, which led to the study of this thesis and that guided its development, was to propose a new and innovative framework in order to help network administrators to configure the network and so to choose automatically which network functions must be enabled to meet the security policies requested by users and to take care to allocate them among the available physical servers.

The *Verifuse* (VERInet FUnction SElection and placement) framework was developed in order to solve this problem. The framework consists of two modules that are:

- *CAID* (CApability IDentifier) which has the aim to analyze the policies, expressed by HPL, to identify which capabilities are needed to satisfy them.
- *SAP* (Selection And Placement) which is intended to determine and select how many and which instances of the network functions are required to meet all the instances of the required capabilities. It has also the task to allocate the instances of the functions between the available physical servers. In fact, the choice of functions is dictated by both the necessary capabilities and the available physical resources.

In the Fig. 4.1 is possible to see what is the workflow, which is summarized in the following points:

1. *CAID* receives in input the repository, containing all the policies to satisfy, and the list which all the capabilities defined;
2. *CAID* analyzes the policies in order to determine which capabilities are necessary;
3. *CAID* sends the set of the capabilities required to another module which has the aim to determine how many instances, for each capability, are necessary;
4. *SAP* receives in input the catalog of all the available network functions, the information on the physical infrastructure and the set of all the instances of the capabilities that are required which has been chosen by another component;
5. *SAP* chooses, optimally, which sub-set of functions is needed in order to meet all the policies and allocates the functions into the physical servers.

Therefore, the framework analyzes the policies expressed in the Policy Repository, which was received as input, by means of the *CAID* module in order to understand what are the capabilities necessary to satisfy all the policies based on the *Capabilities Defined*. At this point, it sends to another component the list of *Capabilities Required*.

Then, it receives, from a module outside the framework, the list of the instances of the capabilities required and it searches between the functions present in the *VNF Catalogue* which one support the requested capabilities and selects the optimal functions, taking into account available physical resources, so as to be able to allocate them in the physical servers. The selection of functions is subject to the conditions imposed, in the first place the functions must be able to support the capabilities but it is also necessary that certain physical resources are available in order to be able to allocate the functions in the servers.In addition, the choice is subject to optimizations, you can choose to reduce the cost of the functions as well as reduce the amount of ram needed and so on.

It should be taken into account that this framework is part of a larger project under publication and that the *PolicyRepository* and the *VNFCatalogue* schemas, as defined, provide a tool to expand and improve the two modules *CAID* and *SAP* in the future.

In order to select the required network functions starting from the definitions of the policies, the framework implements the phase named **Policy Refinement** [14].

The Policy Refinement is the process to determine the resources needed to satisfy policy requirements by translating high-level policies into operational policies. This approach allows the separation between the policy specification and the real configuration of network functions. The administrator can fully focus on policy-making, excluding low-level configuration, because the framework provides him with a high-level view of system behavior.

Therefore, the idea is to analyze the policies, which have been expressed through the HPL, in order to extrapolate the necessary capabilities and then use them to select the network functions to be used.

To achieve these goals, first of all it was necessary to formalize both a model that clearly represents the exhaustive catalogue of security functions and a model that allows network

administrators, even without specific technical knowledge, to express security policies in a simple, unlimited way.

In order to realize the structure of complete and exhaustive models that are easily expandable in the future, with the possibility to customize and constrain every choice, was used the *eXtensible Markup Language (XML)* language[5]. In fact, this language is a markup language and therefore allows the definition of new languages, which can be used to define these new models, also allowing the expression of links and constraints between the different parts.

Although all the XML Schemas of these models will be presented in detail in the Chapter 5, we can now focus on the following concepts:

- *CapabilitiesDefined*, represents all the *Capabilities* supported by the framework;
- *Policy Repository*, allows network administrators to define, in a simple but complete way, all the policies necessary to meet user needs;
- *VNFCatalog*, represents the catalog of the network functions which are available in the system;
- *Hosts*, represents the physical server where it is possible to allocate the instances of network functions.

In particular, the *Policy Repository* allows network administrators to define, in a simple but complete way, all the policies necessary to meet user needs.

For instance, if a father wants to monitor his child by prohibiting him access to social networks or blocking him access to illegal sites, the network administrator must configure the network and then enable all those network functions that are useful for these purposes. The parent may also want to limit the Internet connection only at certain times of the day, so that the son does not distract himself during those afternoon hours that he should devote to the study.

Selecting the necessary network functions in order to properly meet these requests and, at the same time, the other from all users is a very complicated task to be fulfilled by network administrators.

Therefore, the work of administrators is helped and reduced if they have at their disposal a simple way to express policies. So the aim of this model is precisely this, to offer a tool that facilitates as much as possible the writing of these policies.

How can a model simplify the drafting of policies? If the network administrator follows a guided and precise scheme, the definition of these policies turns out to be simpler and manageable.

[5]The XML language, acronym of Extensible Markup Language, is a metalanguage developed by W3C[6] in 1998. As a metalanguage, this language aims to define other languages.

[6]The World Wide Web Consortium (W3C) is an international community, founded in 1994 with the aim to develop open standards to ensure the long-term growth of the Web.

To this purpose, to formalize the scheme, so that it is simple and easy to understand for most end-users (even those who do not have particular technical knowledge), a user-oriented language has been used: the *High-level Policy Language (HLP)* as suggested in [15] and [16].

This language, as better described in the next section, allows administrators to impose their policies without worrying about the low-level configuration part of the network functions. In fact, nowadays the work of administrators is difficult precisely because, besides having to worry about the definition of policies, they also have to take care of the appropriate configuration of the network functions; For instance, the network administrator must properly configure the firewall so that, according to the agreed policies, the packet in transit is intercepted and analyzed (IP source, IP destination, protocol, etc.) for being allowed or not to reach certain web addresses.

The Low-level network function configuration limits the capacity of network administrators as each security function requires a specific language to be configured. The purpose of this model is to divide the two parts so that the network administrator can focus only on the writing of policies without considering how to actually configure the network functions.

Therefore, the network administrator, using this model, can write the policies in an High-level language without worrying about the low-level configuration, which will then be translated using an appropriate tool, fully focusing his attention on policies. In this way there is no need that the network administrator has got particular low level knowledge.

In order to formalize the model of the *Policy Repository*, as suggested in [15] and [16], a new and specific language was used. In the following sections the HPL, the MPL and the LPL will be analyzed in detail.

The thesis more focused on the use of the *High-level Policy Language (HPL)* as a key language to be used by administrators but internally also the *Medium-level Policy Language (MPL)* is used; Furthermore, the model allows experienced network administrators to express policies directly in this last language although the framework currently does not fully support it. It can be considered as a possible future expansion.

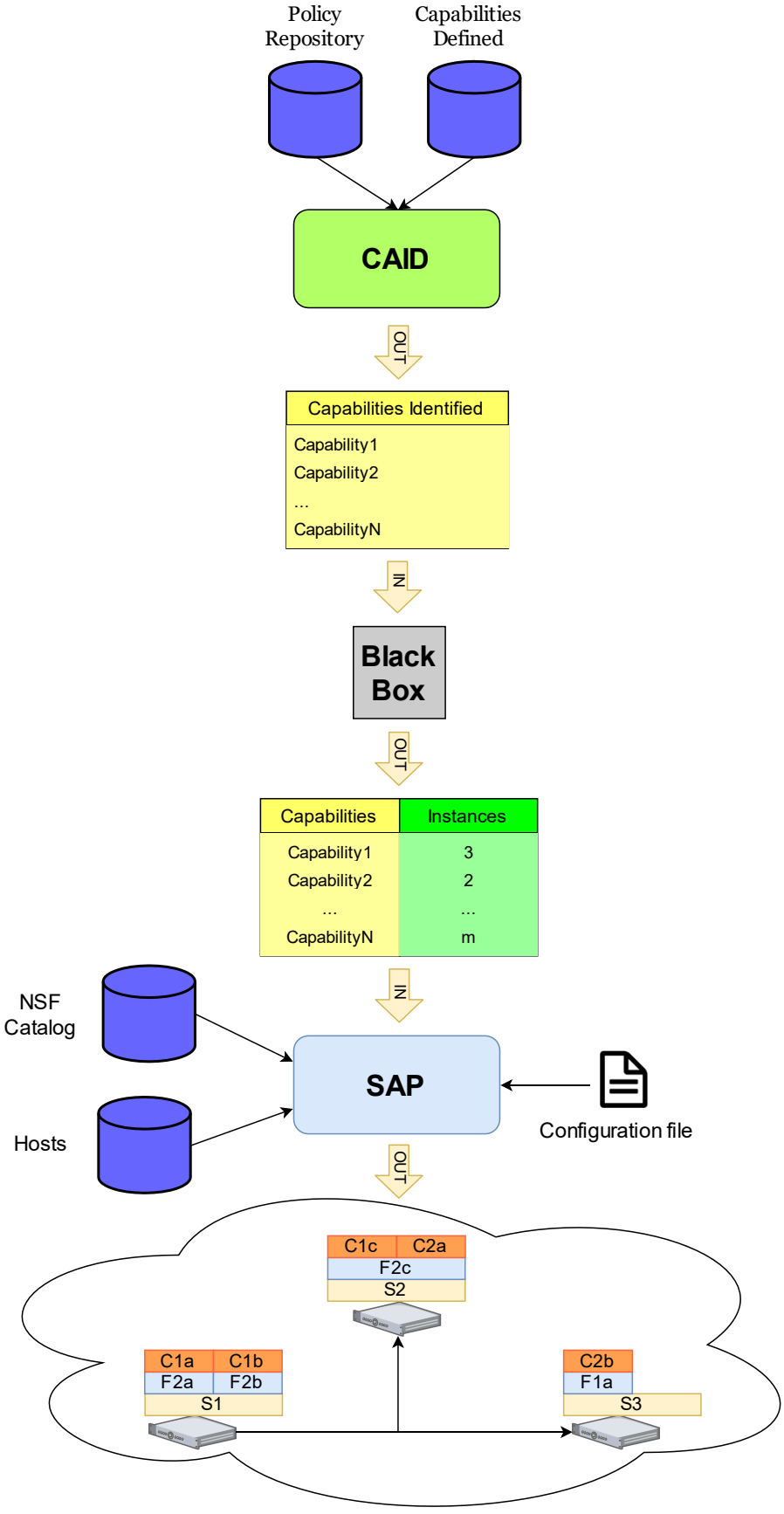

Figure 4.1: Framework's workflow

4.2 Policy Languages

4.2.1 High-level Policy Language (HPL)

Introduction

The High-level Policy Language (HPL) is an authorization language, proposed in the European project *SECURED*[7], with the aim of expressing the user security requirements by hiding the configuration details during the definition of the policies. For this purpose it has been idealized as an user-oriented policy language that exploits sentences close to the natural language.

Therefore, the *HLP* aims to be:

- **Simple** The use of predefined sentences close to the natural language aims to be intuitive even for those users without particular technical knowledge. The end-user appears to be assisted during the writing of the policies by sentences as *"enable antivirus"* or *"Bob is authorized to access Internet"*;
- **Flexible** The model of the *Policy Manager* by exploiting this language results to be very flexible and customizable. It appears to be open to any scenario and there are no particular limits on policy conditions. It is possible to express conditions about time, urls, IP addresses and so on;
- **Extensible** It is easy in the future to extend the model or adapt it to your needs without completely revolutionizing it.

The HPL has played a key role in the thesis; it has been expanded and improved so that the Policy Repository was modeled in the best way.

Structure

The Policy Repository wants to be a model simple and easy to use. At this aim, it is based on the HPL which uses predefined sentences close to the natural language to be immediate. How these sentences are composed?

In the basic version of the HPL of reference the predefined sentences were formulated in this way:

```
[ sbj ] action obj [ (field_type, value) ... (field_type, value) ]
```
(3.1)

Where:

- **sbj** is the user to whom the policy is dedicated and so that needs to access or perform an action on an object; Can be omitted if the subject corresponds to the user that defines the HPL;

[7] SECURED is a project funded by the European Commission with the aim to develop a trusted and virtualized execution environment that allows the users to execute security applications on the network edge device to protect their traffic.

- **action** is the operation (e.g. enable, protect, authorized) that must be performed on the object;
- **obj** is the resource (e.g. Internet, antivirus, DNS, IDS) that is the target of the action;
- **(field_type, value)** is an optional condition (e.g. time, traffic target, content type) that allow to better specify the action on the object.

This is the basic structure, which does not allow a clear and complete description of policies. It has a number of limitations:

- the policy does not have a key which uniquely identifies it;
- it does not allow to correctly specify the difference between author and subject of the policy;
- the concept of the template is not well explained and difficult to use;
- the difference between a specific user or a group of users is not well formulated;
- the meaning and the use of some fields are not well specified, in some cases are abstract information: what is the meaning of traffic target or content type? What is behind it?.

For these reasons the structure has been updated, now is more articulated and it has allowed to model the Policy Repository by different layers.

The key points are as follows:

- each policy is written by an **Author** and is addressed to a **Subject**, which can be a specific user or a user groups;
- each policy consists of a set of rules addressed to the specified subject. These rules take the name of **Policy Rules**;
- each Policy Rule consists of an **Operation** or a **Template** to execute;
- each Template consists of a series of Operations;
- each Operation consists of the **Action** to be performed, the **Object** to which it is addressed and any **Fields** used to specify further conditions.

This model will be presented and analyzed in detail, with examples, in the Chapter 5.

4.2.2 Medium-level Policy Language (MPL)

Introduction

The Medium-level Policy Language (MPL) is an authorization language, which was proposed in the European project *SECURED* like the High-level Policy Language (HPL).

The MPL, unlike the HPL, it is not addressed to end users without technical knowledge but at experienced administrators. It is an abstract language that is still vendor-independent, but allows expressing specific configurations in a generic format. Therefore, these configurations must then be translated into the proprietary language of the device to be configured, so that they are actually used.

Therefore, the MLP aims to describe the configuration settings of groups of devices having some characteristic in common (features that during this thesis have taken the name of *Capabilities* and which will be better described in the next section) and so that can perform typical actions. For example, all firewalls, regardless of vendor, have the task of allowing or blocking the passage of a packet in transit by observing specific fields in the packet, executing precise rules defined by the administrator. But only the rules are actually expressed using the proprietary language which changes from device to device.

The HPL is an authoritative language useful in the description of policies whose use, as it uses sentences very close to human language, is simple and immediate. However, this advantage turns out to be also a disadvantage as it cannot be used directly to configure network functions. They must be converted into an intermediate language (MLP) or directly into a low level language (as LPL).

The MLP, on the other hand, allows to express the same information as HLP, but through operational policies in order to directly exploit this information in order to configure network functions. For this reason, specific technical knowledge is required to be able to use this language. As an instance, the network administrator must know how the firewall works and what information is needed to configure it directly.

Therefore, the MLP allows to express security configurations for function classes independently from the vendor or manufacturer. It is flexible and extensible. It ensures that the policies can be converted from HPL or written directly with this language.

Capability

As mentioned in the previous subsection, the MPL allows to describe the configuration settings of groups of devices having some characteristic in common. These characteristic can be expressed through the idea of *Capability*.

The concept of “Capability” is very abstract and has been very difficult to formalize it during the course of the thesis, it has been a cornerstone of development. This is because the choice of the network functions to satisfy defined policies is driven by the capabilities, so they play a fundamental part.

The Capability represents the intrinsic characteristic of a network function and so precisely and unequivocally clarify what the function can do, how can act and what differentiates it from other functions.

It is important to note that a single VNF can support more functionality and therefore more capability.

For example, if a network function is able to analyze IP packets in transit in a given network node (which can be an endpoint as well as an intermediate node) then this means that it supports the *"Packet_Filter_L4_Statefull"* capability. If the network function can be enabled to act only at certain times in specific days of the week, it also supports the *"Time_Filter"* capability.

As is possible to see in the model defined through the use of the Unified Modelling Language (UML)[8] shown in the Fig. 4.2, in the European project *SECURED* the concept of Capability is very articulate.

In this scenario the model of Capability is hierarchical. There is a root element, called *"Capability"*, which is an abstract class that denotes any capability supported by a class of functions. As an instance, classes can be: traffic filter, spam detector, IDS, IPS, malicious file analyzer and so on. All these classes, or better categories, can be seen in Fig. 4.3. where for each category of *Personal Security Application (PSA)*[9] is done the mapping with the corresponding capabilities.

This idea of the concept of Capability turns out to be so complicated that it is difficult to use it in practice. This thesis aims to develop a framework in which capability must be mapped both with policies contained in the *Policy Repository* and with functions defined in the catalogue *VNFCatalog*. This is because the framework extracts from policies, expressed in HPL, the required capabilities and proceeds looking for those functions that support them.

Therefore, a new model of capability had to be devised. With this aim, the most common network functions were analyzed (e.g., firewall, packet filter, IDS, IPS, etc.), trying to understand how they work at a low level in order to trace the formulation of a model that unites all those functions that, independently from the vendor, belong to a given class.

As an instance, a packet filter level 4[11], can *Allow*, *Block* or *Log* the IP packets in transit and, to decide what action to take, compares the fields: *IP source*, *IP destination*, *Port source*, *Port destination* and *Protocol*; to check if any of these fields fit the rules, written by the administrator who configured the device.

---

[8]The Unified Modelling Language (UML) was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in 1994, it is a modeling language that, in the area of software engineering, aims to provide a standard way to visualize the design of a system [17].

[9]The Personal Security Application (PSA) is a software that is executed on a *Network Edge Device (NED)*[10] . It contains one or more security controls in order to implement security policies and allowing to offload the security from the end-nodes of the network.

[10]The Network Edge Device (NED) is a device which connects a Local Area Network (LAN) to a Wide Area Network (WAN). Therefore, is the first point of the network that is met when an end-node (e.g., personal computer, smart-phone, smart TV) tries to access the internet.

[11]Level 4 means that it works at this layer in the OSI model[12].

[12]The OSI model, aka ISO/OSI Model, is a conceptual framework, proposed by International Organization for Standardization (ISO) and Comité Consultatif International Téléphonique et Télégraphique(CCITT), which imagines the network divided into 7 levels explaining how diverse communication systems can communicate in it using standard protocols. So it aims is the worldwide interoperability [18].

The deep study on the common functions has therefore led to the formulation of a model for the capabilities, which takes into account the intrinsic characteristics of the functions. Therefore, each function class is characterized by a precise capability.
The **Capability model** is so defined:

Model

$$C \stackrel{def}{=} \{f_1, f_2, ..., f_n\} \quad where \quad f_i \neq f_j,\ \forall (i,j) \in \mathbb{N}^2_{1 \to n},\ i \neq j \tag{3.2}$$

- Each Capability C consists of a series of different features f_i, which uniquely identify it;

Constraints

$$C_1 \neq C_2 \tag{3.2.1}$$

- Each capability is unique, there can not be two capabilities consisting of the same set of features;

$$C_1 \not\subset C_2 \tag{3.2.2}$$

- The capabilities are unique and cannot be completely included in each other. This means that the feature set is unique for each capability. Different capabilities may have common features, but may not include the whole set;

$$C_1 \neq C_2 \cup C_3 \tag{3.2.3}$$

- Each capability is unique, Each capability is unique, can not be achieved by combining different capabilities;

Since capabilities have been defined to reflect a specific class of functions, each network function, which is present in the *VNF Catalog* function catalogue, must support one or more capabilities.

The capabilities have been chosen so as to be perfectly compatible also with the HPL model that allows to express security policies through the *Policy Repository*.

An example of capabilities is shown below, the full list will be shown in the Chapter 5 as it was thought.

```
<Capabilities>
   <Capability CapabilityID=‘‘Packet_Filter_L4_Statefull’’>
      <Features>
         <Feature>Packet_Filter</Feature>
         <Feature>L4</Feature>
         <Feature>Statefull</Feature>
      </Features>
   </Capability>

   <Capability CapabilityID=‘‘Packet_Filter_L4_Stateless’’>
      <Features>
         <Feature>Packet_Filter</Feature>
         <Feature>L4</Feature>
         <Feature>Stateless</Feature>
      </Features>
   </Capability>
</Capabilities>
```

As is possible to see in the instance, there are present two capabilities:

1. C_1 = {*Packet_Filter, L4, Statefull*}, which is a *Packet_Filter_L4_Statefull*.
2. C_2 = {*Packet_Filter, L4, Stateless*}, which is a *Packet_Filter_L4_Stateless*;

These two capabilities describe the behaviour of a generic *Packet Filter*, which works at level 4 of the ISO/OSI protocol stack. The difference between the two *Packet Filter* lies in the fact that C_1 supports connection status, so it can track of the state of network connections (such as TCP streams or UDP communication), while C_2 does not.

Inasmuch as the two set di features contain the two common feature (Packet_Filter, L4) but the last one is different (*Statefull* for C_1 and *Stateless* for C_2), the constraints (3.2.1), (3.2.2) and (3.2.3) are respected, as the two capabilities are unique as the features set in them are different and the one does not completely include the other.

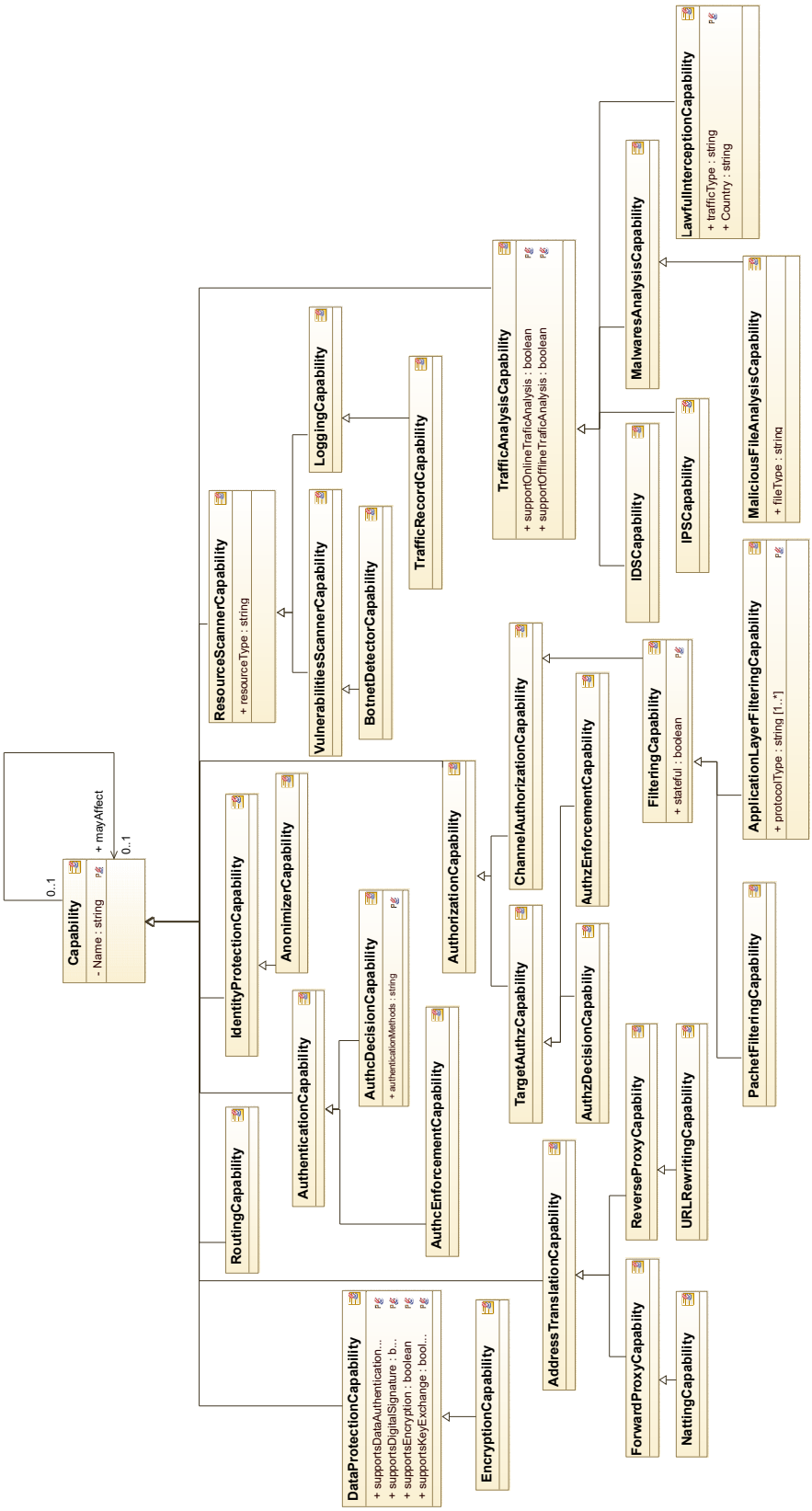

Figure 4.2: Capability hierarchy in SECURED

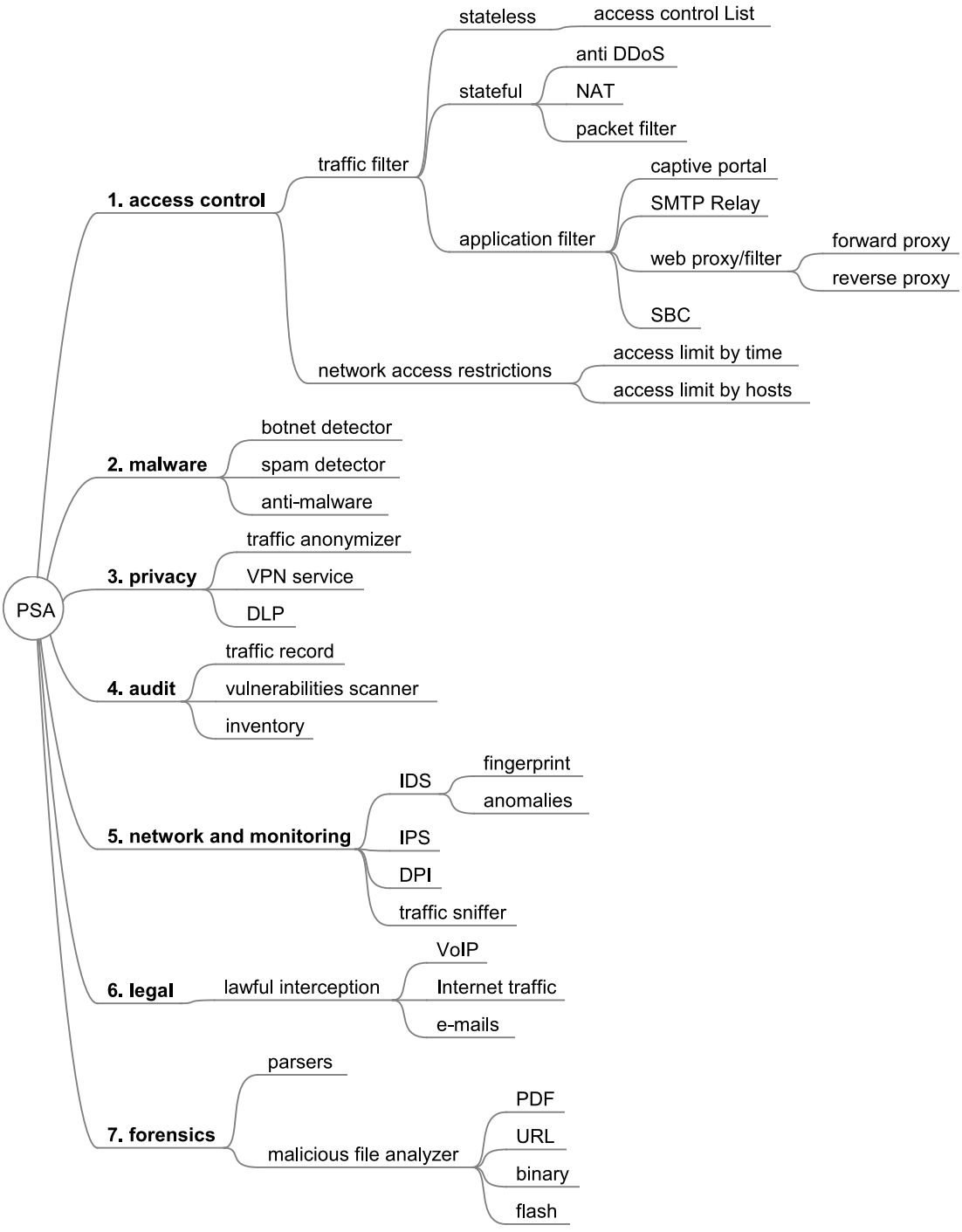

Figure 4.3: Categories of PSA in SECURED

4.2.3 Low-level Policy Language (LPL)

The *Low-level Policy Language (LPL)* represents the proprietary language that must be manually configured in the devices. In fact, HPL is a high-level language that has been created to allow administrators to focus on policy-making without taking into account how these are actually implemented by the various functions of the network. Similarly, MLP allows administrators to configure network devices (for example, by configuring firewalls to block or forward IP packets that are in transit) but always in a generic way that is common to a given device class but without going into actual configuration details. Therefore, HLP and MLP are languages that cannot be used to actually configure devices (as an instance, a "Cisco" router needs instructions in its own language to be configured). It is for these reasons that a second phase is necessary in which these medium and high level languages are converted into proprietary languages and therefore low level languages.

This aim, however, appears to be outside the context of this thesis since the network functions, which are selected and selected to meet policy, are software functions.

4.3 Capability Identification

As we have seen in the previous paragraphs, the CAID module, which has been developed, receives in input the *Policy Repository* and the list of *Capabilities Defined* in the framework producing as output the list of *Capabilities Required* to satisfy all the policies.

The *CAID* module does the *Capability Identification* starting the process of *Policy Refinement*, whose overall purpose is to determine the network functions needed to satisfy all the policy requirement.

Therefore, the *CAID* module analyzes all the policies present in the policy repository, which have been expressed through the *HPL*, and decides which capabilities are necessary to satisfy them.

The module, to extrapolate capabilities correctly, uses an internal mapping consisting of a map which associates the *Operation* expressed by the policy with the relative *Capacity* that supports it. In particular, the framework analyzes the Operation in order to verify which Action to execute, on which Object and if there are some Field to hold in consideration. This is because the map associates both each <Action, Object> pair and each Field with the respective capability.
As an instance,

```
<Action, Object> : <AUTHORIZED_ACCESS, VO_IP_TRAFFIC>
        is mapped to:  Capability: FW_Application
```

while

```
the Field: TIME_PERIOD
is mapped to:  Capability: Time_Filter
```

The complete list is shown below, in the Tab. 4.1 and the Tab. 4.2.

In this way the framework is able to identify all the capabilities that are needed to meet all the policies correctly. At this point it is necessary to select, and therefore to choose, which functions contained in the catalogue of the network functions, to allocate.

Field	Capability
TIME_PERIOD	Time_Filter
TRAFFIC_TARGET	Packet_Filter_L4_Statefull
SPECIFIC_URL	FW_Web_Application
TYPE_OF_CONTENT	Packet_Filter_L4_Statefull
PURPOSE	Antivirus
BANDWIDTH	BandwidthManagement

Table 4.1: The map between Field and Capability

Action	Object	Capability
NOT_AUTHO-RIZED_ACCESS	VO_IP_TRAFFIC	FW_Application
AUTHORIZED_AC-CESS	VO_IP_TRAFFIC	FW_Application
NOT_AUTHO-RIZED_ACCESS	P_2_P_TRAFFIC	FW_Application
AUTHORIZED_AC-CESS	P_2_P_TRAFFIC	FW_Application
NOT_AUTHO-RIZED_ACCESS	TRAFFIC_3_G	FW_Application
AUTHORIZED_AC-CESS	TRAFFIC_3_G	FW_Application
NOT_AUTHO-RIZED_ACCESS	TRAFFIC_4_G	FW_Application
AUTHORIZED_AC-CESS	TRAFFIC_4_G	FW_Application
NOT_AUTHO-RIZED_ACCESS	INTERNET_TRAFFIC	Packet_Filter_L4_State-full
AUTHORIZED_AC-CESS	INTERNET_TRAFFIC	Packet_Filter_L4_State-full
NOT_AUTHO-RIZED_ACCESS	INTRANET_TRAFFIC	Packet_Filter_L4_State-full
AUTHORIZED_AC-CESS	INTRANET_TRAFFIC	Packet_Filter_L4_State-full
NOT_AUTHO-RIZED_ACCESS	DNS_TRAFFIC	Packet_Filter_L4_State-full
AUTHORIZED_AC-CESS	DNS_TRAFFIC	Packet_Filter_L4_State-full
NOT_AUTHO-RIZED_ACCESS	DNS_REQUEST	Packet_Filter_L4_State-full
AUTHORIZED_AC-CESS	DNS_REQUEST	Packet_Filter_L4_State-full
NOT_AUTHO-RIZED_ACCESS	DNS_RESPONSE	Packet_Filter_L4_State-full
AUTHORIZED_AC-CESS	DNS_RESPONSE	Packet_Filter_L4_State-full
NOT_AUTHO-RIZED_ACCESS	ALL_TRAFFIC	Packet_Filter_L4_State-full
AUTHORIZED_AC-CESS	ALL_TRAFFIC	Packet_Filter_L4_State-full
ENABLE	FILE_SCANNING	FileScanning
ENABLE	EMAIL_SCANNING	EmailScanning
ENABLE	ANTIVIRUS	Antivirus

ENABLE	IDS	IDS
ENABLE	IPS	IPS
ENABLE	D_DOS_ATTACK_PRO-TECTION	DDosAttackProtection
ENABLE	LOGGING	Logging
ENABLE	BA-SIC_PARENTAL_CON-TROL	Parental_Control
ENABLE	AD-VANCED_PARENTAL_C(TROL	Parental_Control
REMOVE	PUBLIC_IDENTITY	IdentityProtection
REMOVE	TRACKING_TECH-NIQUES	BlockTracking
REMOVE	ADVERTISEMENT	BlockAdvertisement
REDUCE	BANDWIDTH	BandwidthManagement
CHECK_OVER	SECURITY_STATUS	Antivirus
COUNT	DNS_TRAFFIC	Packet_Filter_L4_State-full
COUNT	DNS_REQUEST	Packet_Filter_L4_State-full
COUNT	DNS_RESPONSE	Packet_Filter_L4_State-full
COUNT	CONNECTION	Packet_Filter_L4_State-full
PROTECT_CONFIDEN-TIALITY	VO_IP_TRAFFIC	ProtectConfidentiality
PROTECT_CONFIDEN-TIALITY	P_2_P_TRAFFIC	ProtectConfidentiality
PROTECT_CONFIDEN-TIALITY	TRAFFIC_3_G	ProtectConfidentiality
PROTECT_CONFIDEN-TIALITY	TRAFFIC_4_G	ProtectConfidentiality
PROTECT_CONFIDEN-TIALITYS	INTERNET_TRAFFIC	ProtectConfidentiality
PROTECT_CONFIDEN-TIALITY	INTRANET_TRAFFIC	ProtectConfidentiality
PROTECT_CONFIDEN-TIALITY	DNS_TRAFFIC	ProtectConfidentiality
PROTECT_CONFIDEN-TIALITY	DNS_REQUEST	ProtectConfidentiality
PROTECT_CONFIDEN-TIALITY	DNS_RESPONSE	ProtectConfidentiality
PROTECT_CONFIDEN-TIALITY	ALL_TRAFFIC	ProtectConfidentiality

PROTECT_INTEGRITY	VO_IP_TRAFFIC	ProtectIntegrity
PROTECT_INTEGRITY	P_2_P_TRAFFIC	ProtectIntegrity
PROTECT_INTEGRITY	TRAFFIC_3_G	ProtectIntegrity
PROTECT_INTEGRITY	TRAFFIC_4_G	ProtectIntegrity
PROTECT_INTEGRITY	INTERNET_TRAFFIC	ProtectIntegrity
PROTECT_INTEGRITY	INTRANET_TRAFFIC	ProtectIntegrity
PROTECT_INTEGRITY	DNS_TRAFFIC	ProtectIntegrity
PROTECT_INTEGRITY	DNS_REQUEST	ProtectIntegrity
PROTECT_INTEGRITY	DNS_RESPONSE	ProtectIntegrity
PROTECT_INTEGRITY	ALL_TRAFFIC	ProtectIntegrity

Table 4.2: The map between <Action:Object> and Capability

4.4 Selection, Optimization and Placement

The SAP module receives in input the list containing for each capability the number of its instances that are necessary and also receives the catalogue of network functions available in the system (*VNF catalog*) together with the list of usable physical hosts. It also receives a configuration file that contains the settings that will be used in the selection and optimization phase. It contains, for example, the priorities with which the resources must be optimized: is it necessary to choose the network functions in such a way as to reduce the cost or the cpu used?

The aim of the module is to select the best subset of functions, among those available, according to the chosen optimization policy (e.g. fewer functions used, reduce the consumption of CPU and RAM used, reduce the Disk and bandwidth required, etc.) so as to satisfy all the required capabilities. This is possible because each function, defined in the catalog, supports one or more capabilities. So the purpose of the module is first to identify which functions support the necessary capabilities and, among them, to choose a subset. The selection of this subset is part of the optimization phase, the module tries to choose the best subset according to some criteria.

As an instance: use fewer functions, minimize the cost, minimize the use of ram and so on.

Actually the module does not simply select the functions but decides how many instances, of the function in question, are necessary.

As an instance: if two instances of C1 capability are to be supported and if the F1 function supporting C1 is present in the function catalogue, then F1 will need two instances.

After selecting the necessary instances of functions, the framework proceeds with their actual allocation in the physical servers available. Actually these two phases (choice of functions and their placement) are executed at the same time. This is because the choice of functions is dictated by the availability of physical servers. For example, a given function is not selected if there is no physical server available with adequate ram capacity.

In order to choose which network functions to allocate, first of all it is necessary that the functions available implement at least one of the necessary capabilities. If there are more functions in the catalogue that implement the same capabilities then the choice becomes difficult and almost impossible to do ”by hand” or in ”static” mode, it should be noted that since you have a catalog then you can also have hundreds of functions to choose from and you also have to be able to place functions between the available servers.

To overcome this problem, it was decided to use the ILP tool and in particular the **Gurobi Solver**. Initially the SMT solver z3, offered by Microsoft, had been used but then everything was converted into ILP in order to use Gurobi.

All formulas will still be shown in the Chapter 6

Gurobi Solver, of all his functions, is able to solve mathematical model whose requirements are represented by linear relationships, how can this feature be used to decide which

functions to use and where to allocate them?

This is possible if to every instance of the network function we associate a binary variable that assumes value 1 if it is used and value 0 in otherwise case. The same thing can be done for hosts and for relations between both *Capability* and *VNF* and *VNF* and *hosts*: it is used a new variable that is 1 if the function is allocated to that host and 0 otherwise.

As an instance, if we have a scenario composed by the following elements:

- $C1_1$ and $C1_2$: two instances of the capability *C1*;
- *VNF1*: VNF that support the capability *C1*;
- *HostA*: physical host available in the system;

We can consider the following binary variables:

- $x^c_{1,1} = \begin{cases} 1 & \text{if } C1_1 \text{ is used} \\ 0 & \text{otherwise} \end{cases}$
- $x^c_{1,2} = \begin{cases} 1 & \text{if } C1_2 \text{ is used} \\ 0 & \text{otherwise} \end{cases}$
- $x^n_{1,1} = \begin{cases} 1 & \text{if } VNF1_1 \text{ is used} \\ 0 & \text{otherwise} \end{cases}$
- $x^n_{1,2} = \begin{cases} 1 & \text{if } VNF1_2 \text{ is used} \\ 0 & \text{otherwise} \end{cases}$
- $y^{1,1}_{1,1} = \begin{cases} 1 & \text{if } C1_1 \text{ is assigned to } VNF1_1 \\ 0 & \text{otherwise} \end{cases}$
- $y^{1,2}_{1,1} = \begin{cases} 1 & \text{if } C1_1 \text{ is assigned to } VNF1_2 \\ 0 & \text{otherwise} \end{cases}$
- $y^{1,1}_{1,2} = \begin{cases} 1 & \text{if } C1_2 \text{ is assigned to } VNF1_2 \\ 0 & \text{otherwise} \end{cases}$
- $y^{1,2}_{1,2} = \begin{cases} 1 & \text{if } C1_2 \text{ is assigned to } VNF1_2 \\ 0 & \text{otherwise} \end{cases}$
- $z^1_{1,1} = \begin{cases} 1 & \text{if } N1_1 \text{ is assigned to } HostA \\ 0 & \text{otherwise} \end{cases}$
- $z^1_{1,2} = \begin{cases} 1 & \text{if } N1_2 \text{ is assigned to } HostA \\ 0 & \text{otherwise} \end{cases}$

Now that the variables have been created, Gurobi can proceed. The task of the tool is to assign to each variable the value 0 or 1, respecting certain constraints imposed and trying to optimize the choice according to the parameters defined by the user.

As an instance, starting from the following formula:

$$y_{1,1}^{1,1} + y_{1,1}^{1,2} = 1; \tag{4.1}$$

Gurobi tries to assign the values to the variables making sure that only one of the two variables assumes value 1. This is because every instance of capability must be assigned to a single instance of function.

The notation used and the complete list of defined symbols and constraints regarding *Gurobi* will be discussed and analyzed in detail in the Chapter 6. Now we can introduce the argument by showing the **Configuration file**:

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3  <properties>
4      <!-- Paths where the xml files are located -->
5      <entry key="pathXMLPolicyRepository">xsd/PolicyRepository.xml</entry>
6      <entry key="pathXMLVNFCatalog">xsd/VNFCatalog.xml</entry>
7      <entry key="pathXMLHosts">xsd/hosts.xml</entry>
8
9      <!-- Configuration parameters concerning the use of a function -->
10     <entry key="VNFmustBeSupportedByHost">true</entry>
11
12     <!-- Configuration parameters concerning the optimization of resources
           ↪ -->
13     <entry key="cpu">10</entry>
14     <entry key="ram">20</entry>
15     <entry key="disk">5</entry>
16     <entry key="bandwidth">2</entry>
17     <entry key="cost">30</entry>
18 </properties>
```

The *Configuration file* contains a number of useful information that will then be used to express constraints and guide optimization choices. The file is in XML format, so the framework can access the variables as properties.

The entries in the file are:

- **pathXMLPolicyRepository**, string variable, represents the path on the disk where the *Policy Repository* is present;
- **pathXMLVNFCatalog**, string variable, represents the path on the disk where the *VNFCatalog* is present;

- **pathXMLHosts**, string variable, represents the path on the disk where the *Hosts* are present;
- **VNFmustBeSupportedByHost**, boolean variable, it is true if the host must support the function in order to allocate it.

  For example, if you need to allocate the N1 function and physical hosts H1 and H2 are available but only H1 supports N1 then, if the variable is true, only the latter can allocate it otherwise both can allocate it;
- **cpu**, integer variable, expresses CPU priority during resource optimization;
- **ram**, integer variable, expresses ram priority during resource optimization;
- **disk**, integer variable, expresses disk priority during resource optimization;
- **bandwidth**, integer variable, expresses bandwidth priority during resource optimization;
- **cost**, integer variable, expresses cost priority during resource optimization;

The priority given to each resource is very important during the optimization phase.

This is because the framework supports the multi-objectives: it seeks the best solution obtainable by optimizing the resource with maximum priority but, in case there are more solutions with this resource then the framework tries to optimize the other resources and so on, everything hierarchically.

4.5 Example

As an instance, we can consider the scenario composed by the following element:

- **Policy1**:

```
<PolicyHPL PolicyID="Policy1">
    <Author>Admin</Author>
    <PolicyRuleHPL>
        <Subject>
            <SpecificUser>User 1</SpecificUser>
        </Subject>
        <Operation>
            <Action>authorized_access</Action>
            <Object>VoIP_traffic</Object>
        </Operation>
    </PolicyRuleHPL>
</PolicyHPL>
```

The *Admin* who is the *Author* of the *Policy1* wrote a *PolicyRule* addressed to the *User1* with the aim to *authorized* him to use *VoIP traffic*.

- **Policy2**:

```
<PolicyHPL PolicyID="Policy2">
    <Author>Admin</Author>
    <PolicyRuleHPL>
        <Subject>
            <SpecificUser>User2</SpecificUser>
        </Subject>
        <Operation>
            <Action>enable</Action>
            <Object>email_scanning</Object>
        </Operation>
    </PolicyRuleHPL>
</PolicyHPL>
```

The *Admin* who is the *Author* of the *Policy2* wrote a *PolicyRule* addressed to the *User2* with the aim to *enable* his *email scanning*.

- **Policy3**:

```
<PolicyHPL PolicyID="Policy3">
    <Author>Admin2</Author>
    <PolicyRuleHPL>
        <Subject>
            <SpecificUser>User2</SpecificUser>
        </Subject>
        <Template>Template1</Template>
    </PolicyRuleHPL>
</PolicyHPL>
```

  The *Admin2* who is the *Author* of the *Policy3* wrote a *PolicyRule* addressed to the *User2* with the aim to *execute* the *Template1*, which *enables email scanning and antivirus*.

Therefore, exploiting the mapping described in the Tab. 4.2 and Tab. 4.1 we can say that:

- to satisfy the *Policy1* we need the Capability *FW_Application* because it maps the tupla *AUTHORIZED_ACCESS : VO_IP_TRAFFIC*;
- to satisfy the *Policy2* we need the Capability *EmailScanning*, which supports the couple *ENABLE : EMAIL_SCANNING*;
- to satisfy the *Policy3* we need the Capabilities *EmailScanning and Antivirus* which implement the related actions.

As it is possible to see in the Fig. 4.4.

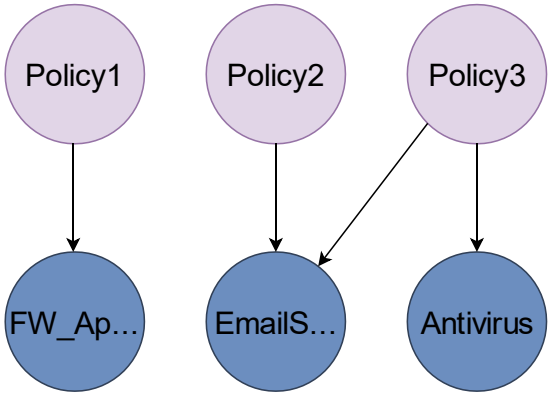

Figure 4.4: Example: Three Policies with four Capabilities

To support these capabilities, the NSFs catalog consists of a series of functions, including the following, which support the above defined capabilities:

- **NSF1**:

```
<NSF NSF_ID="NSF1">
    <HardwareInfo>
        <RAM value="4" unit="GB"/>
        <Disk value="4" unit="GB"/>
        <Cost>2</Cost>
    </HardwareInfo>
    <Functionality>
        <CapabilitiesRef>
            <CapabilityRef CapabilityID="Antivirus"/>
            <CapabilityRef CapabilityID="EmailScanning"/>
        </CapabilitiesRef>
    </Functionality>
</NSF>
```

The *NSF1* requires 4GB of RAM, 4 GB of Disk and coast 2. It is able to support the capabilities *Antivirus* and *EmailScanning*;

- **NSF2**:

```
<NSF NSF_ID="NSF2">
    <HardwareInfo>
        <RAM value="6" unit="GB"/>
        <Disk value="2" unit="GB"/>
        <Cost>4</Cost>
    </HardwareInfo>
    <Functionality>
        <CapabilitiesRef>
            <CapabilityRef CapabilityID="Antivirus"/>
            <CapabilityRef CapabilityID="EmailScanning"/>
        </CapabilitiesRef>
    </Functionality>
</NSF>
```

The *NSF2* requires 6GB of RAM, 2 GB of Disk and coast 4. It is able to support the capabilities *Antivirus* and *EmailScanning*;

- **NSF3**:

```
<NSF NSF_ID="NSF3">
    <HardwareInfo>
        <RAM value="2" unit="GB"/>
        <Disk value="4" unit="GB"/>
        <Cost>4</Cost>
    </HardwareInfo>
    <Functionality>
        <CapabilitiesRef>
            <CapabilityRef CapabilityID="FW_Application"/>
        </CapabilitiesRef>
    </Functionality>
</NSF>
```

The *NSF3* requires 2GB of RAM, 4 GB of Disk and coast 4. It is able to support the capability *FW_Application*.

As it is possible to see better in the Fig. 4.5.

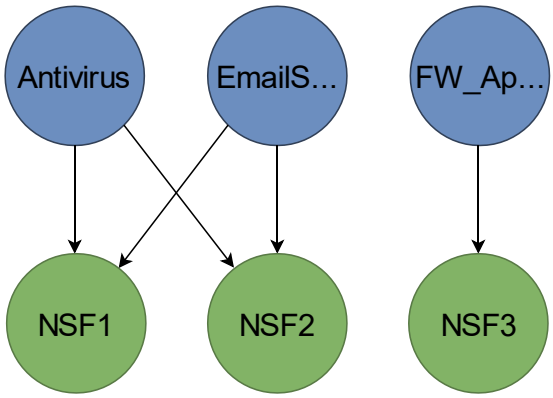

Figure 4.5: Example: Four Capabilities supported by three NSFs

It is also necessary that the functions, or rather their instances, are physically located among the physical servers available.
The following hosts are available for this purpose:

- **host1**:

```
<Host HostID="host1" cpu="10" ram="30000" disk="300000" bandwidth
    ↪ ="100">
    <SupportedVNF>NSF1</SupportedVNF>
    <SupportedVNF>NSF3</SupportedVNF>
</Host>
```

The *host1* has at is disposal *10 Ghz of Cpu*, *30 GB of Ram*, *300 GB of Disk* and *100 Mb/s of Bandwidth* at most.

- **host2**:

```
<Host HostID="host2" cpu="50" ram="60000" disk="500000" bandwidth
    ↪ ="80">
    <SupportedVNF>NSF1</SupportedVNF>
    <SupportedVNF>NSF2</SupportedVNF>
</Host>
```

The *host2* has at is disposal *50 Ghz of Cpu*, *60 GB of Ram*, *500 GB of Disk* and *80 Mb/s of Bandwidth* at most.

- **host3**:

```
<Host HostID="host3" cpu="30" ram="8000" disk="50000" bandwidth="
    ↪ 50">
    <SupportedVNF>NSF1</SupportedVNF>
    <SupportedVNF>NSF2</SupportedVNF>
    <SupportedVNF>NSF3</SupportedVNF>
</Host>
```

The *host3* has at is disposal *30 Ghz of Cpu*, *8 GB of Ram*, *50 GB of Disk* and *50 Mb/s of Bandwidth* at most.

As it is possible to see in the Fig. 4.6.

Figure 4.6: Example: Two Host

Now, in this scenario, the question is: how many and which instances of functions are needed to meet capabilities? Where do you physically allocate them?

In order to be able to answer these questions first of all it is necessary to note that there are conditions that must be fulfilled: the *Hard-Constraints*, which will be explained in detail in the Chapter 6.

The key points are as follows:

1. In order to satisfy the policy one or more capabilities are necessary and so their instances;

2. All the capabilities required must be supported;

3. For each capability instance a function instance is required;
4. A single function instance can support multiple instances of capabilities if they refer to different capabilities;
5. A single function instance is used if and only if there is an instance of capability which use it;
6. Each host can support different function instances.
7. Each host is used if and only if there is an instance of function which use it.

Therefore:

- The *Policy1* requires the Capability *FW_Application*;
- The *Policy2* requires the Capability *EmailScanning*;
- The *Policy3* also requires the Capability *EmailScanning* and the Capability *Antivirus*.

It is important to keep in consideration that the developed framework for this thesis, does not include the part of "Allocation and Distribution" which is carried out by the *Verefoo* framework. For this reason, the list of the instances of the capabilities to be allocated is given by the other module. This is because Verefoo is aware of the network infrastructure and therefore can decide that a given capacity is needed at multiple points of the network and therefore request an additional instance.

At this point, imagining that from Verifoo we received the following list of capability instances:

- *FW_Application*: 1 instance;
- *EmailScanning*: 2 instances;
- *Antivirus*: 1 instances;

Therefore, as it is possible to see in the Fig. 4.7, the instances of the functions which could be allocated are the following:

- **NSF1_1** and **NSF1_2**: the function *NSF1* could theoretically require two instances because there are two instances of the same capability (*EmailScanning*) and each requires a different instance of the function to be allocated. The function *NSF1* supports also the capability *Antivirus* which can be allocated in one of the two instances;
- **NSF2_1** and **NSF2_2**: the function *NSF2* could theoretically require two instances for the same reason as *NSF1*;
- **NSF3_1**: the function *NSF3* requires one instance because it supports only the *FW_Application* of which only one instance is required.

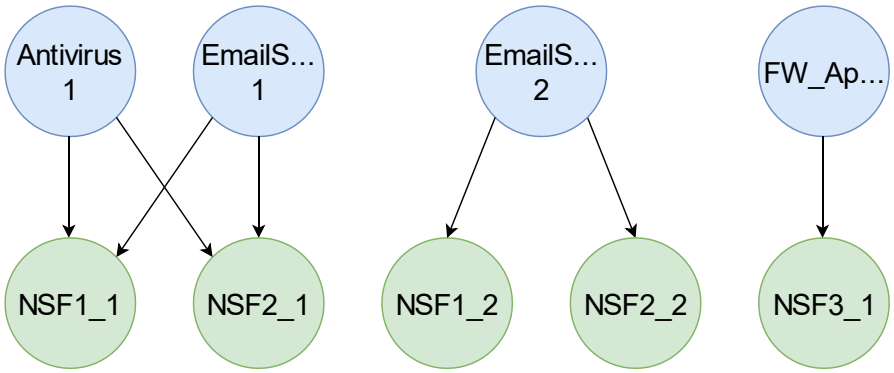

Figure 4.7: Example: Capabilities instances with instances of NSFs which could allocate them

It is important to note that functions *NSF1* and *NSF2* require theoretically two instances each, because three capabilities are required to be allocated of which only two are in conflict (instances of the same capability) but we don't know where they will come allocated even if in practice a maximum of three are required.

Which of the available instances will be chosen? This decision depends on the soft constraints that you want to impose. For example, do you want to optimize the consumption of ram or cpu? Or the cost of the function?

The selection of functions also depends on another factor, namely the amount of physical servers that are available. In fact for a function to be selected it is necessary that it is allowable in a physical server, therefore there must be a host that can support it and that has adequate capacities in terms of available resources.

Since the choice of functions is closely connected to the capabilities and hosts, it is executed taking into account both so much that this phase takes the name of *"Selection and Placement"*.

Therefore, the scenario to be optimized is the one shown in the Fig. 4.8.

At this point we can proceed with the selection and placement of functions. If we plan to optimize the cost of the functions, the solution is the one shown in the Fig. 4.9, where between the functions *NSF1* and *NSF2* has been chosen the *NSF1* which has a lower cost.

While if we wanted to optimize the consumption of ram the solution would be the one reported in the Fig. 4.10, where the function *NSF2* has been chosen instead of the *NSF1* but at the expense of the other resources, in fact it has been allocated, besides the *host1*, also the *host2* because the first does not support the function *NSF2*.

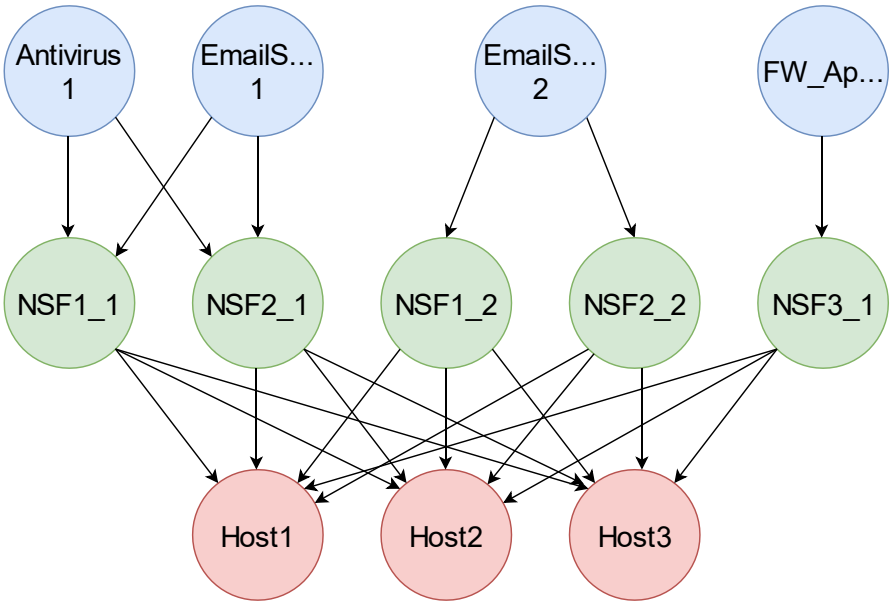

Figure 4.8: Example: Capabilities instances with instances of NSFs which could allocate them and hosts available

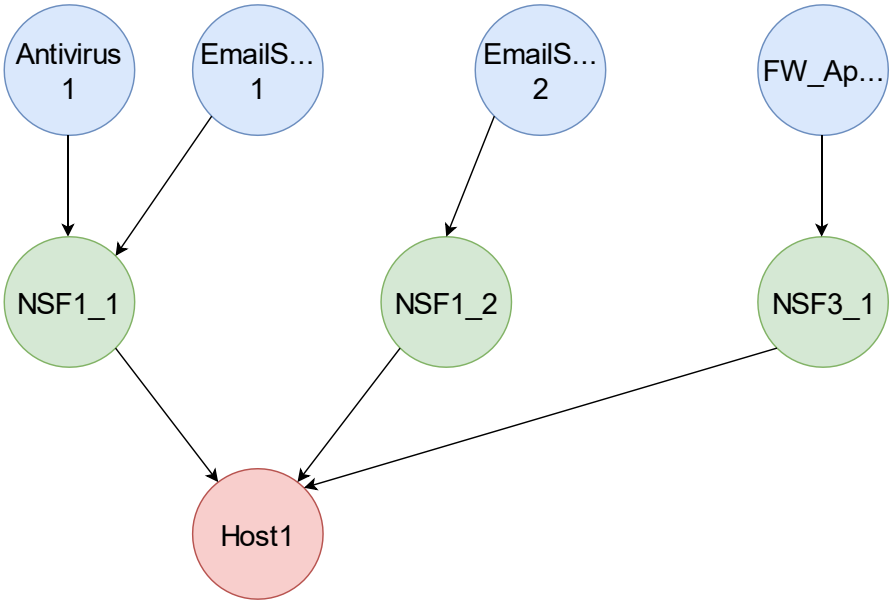

Figure 4.9: Example: Solution 1

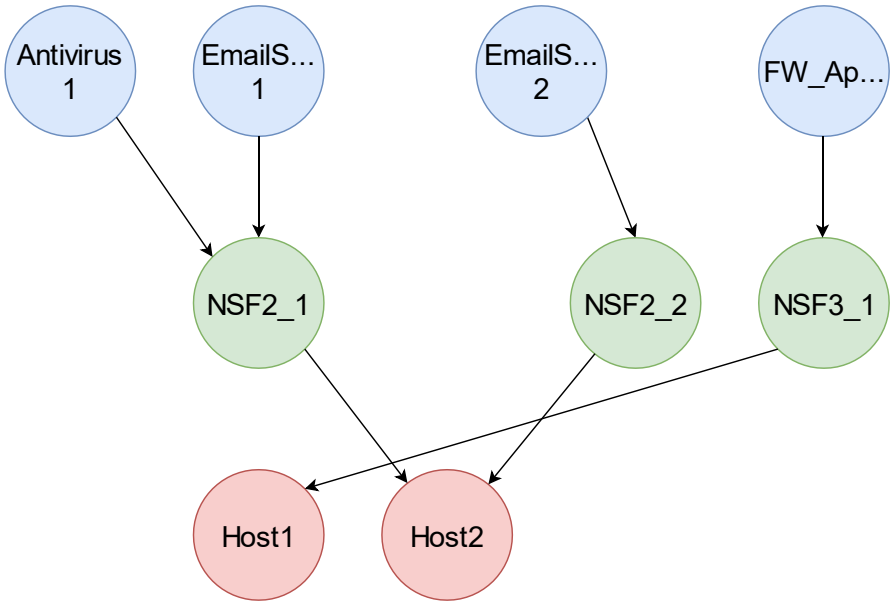

Figure 4.10: Example: Solution 2

Chapter 5

Policy and Function Models

In this chapter we will describe and analyze the XML files that serve as a model for the *Policy Repository*, for the *NSF Catalog*, for the *Capability* and for the *Hosts*. Some use cases will also be shown.

5.1 Policy Repository

The *Policy Repository* allows network administrators to define, in a simple but complete way, all the policies necessary to meet user needs.

As extensively described in chapter 3, to better model the policy repository we followed the approach taken in the European project *SECURED* and therefore formalized and modelled it using a high-level language that is easy to understand for end users: the *HPL*. To this end, we started from the basic version described in the European project and expanded, making it more general and filling it with information.

The fundamental point was to divide it by levels, there is no longer a single expression as in the formula 3.1. Although the "heart" remained this, the formulation of the model has changed completely in order to include additional information in a simple way.

The *root element* is the **"PolicyRepository"**.

```
1  <element name="PolicyRepository">
2      <complexType>
3          <sequence>
4              <element name="KnowledgeBase"
5                  type="tns:KnowledgeBase" minOccurs="0" maxOccurs="1"/>
6              <element name="PolicyRepositoryHPL"
7                  type="tns:PolicyRepositoryHPL" minOccurs="0" maxOccurs="1"/
                       ↪ >
8              <element name="PolicyRepositoryMPL"
9                  type="tns:PolicyRepositoryMPL" minOccurs="0" maxOccurs="1"/
                       ↪ >
10         </sequence>
11     </complexType>
12 </element>
```

It is composed by:

- the *KnowledgeBase*, which is a common database;
- the *PolicyRepositoryHPL*, which contains the real policies expressed by the High-level Policy Language;
- the *PolicyRepositoryMPL*, which is an empty element and has been inserted to allow compatibility for future expansions. In fact, the current framework does not manage the MPL language but it lays the foundations for doing so.

Let's go see all the elements.

5.1.1 KnowledgeBase

```
1  <complexType name="KnowledgeBase">
2      <sequence>
3          <element name="BlackList"
4                 type="tns:BlackList" minOccurs="0" maxOccurs="1"/>
5          <element name="TrafficTargets"
6                 type="tns:TrafficTargets" minOccurs="0" maxOccurs="1"/>
7          <element name="Users"
8                 type="tns:Users" minOccurs="1" maxOccurs="1"/>
9      </sequence>
10 </complexType>
```

The *KnowledgeBase* is a kind of database which contains all the knowledge that is shared and that can be exploited by all the policies.

It is composed by:

- the *BlackList*, which contains information about all the sites, domains, urls and ports which must be blocked;
- the *TrafficTargets*, which contains information on all the IP addresses which constitute a target to be blocked;
- the *Users*, are the users of the network, may be the authors of the policies or users to whom they are addressed.

5.1.2 BlackList

```
<complexType name="BlackList">
    <sequence>
        <element name="Association"
               type="tns:Association" minOccurs="0" maxOccurs="unbounded"/
                  ↪ >
    </sequence>
</complexType>
```

The *BlackList* consists of a series of *Associations*, each of which is characterized by a unique identifier: the *AssociationID*.

The idea is to identify a group of sites, whose domain or url is known, having a feature in common so that you can identify them with a keyword. By doing so, it is possible to block or allow certain categories of addresses in a simple way.

For example, you can define the association "social networks" and associate it with the urls of all social networks (such as Facebook, Twitter, Linkedin, etc.) that you want to block.

In this way, the network administrator can easily help the parent who wants to deny the child access to these sites at certain times. It is necessary that he compiles a policy for him in which "does not authorize access to social networks in certain hours" and automatically connects with all registered sites under "social network".

In addition to domains and urls, you can also associate the identifier to the ports. In this way the administrator can decide to block all those services that try to access the specified port and this simplifies and makes it safer to control certain traffic flows [19]. As an instance, it is possible to block all http traffic by blocking port 80.

An important thing to consider is that this database of associations can be dynamic and therefore can be thought of as a remote repository that will update automatically. In fact there are some software, including "E2Guardian", that offer these common repositories to download and update. This is certainly an advantage as it is common to all inexorably richer in information and also takes away from administrators the task of creating it.

This is the scheme of the Association:

```
<complexType name="Association">
    <sequence>
        <element name="Domains"
               type="tns:Domains" minOccurs="1" maxOccurs="1"/>
        <element name="URLS"
               type="tns:URLS" minOccurs="1" maxOccurs="1"/>
        <element name="Ports"
               type="tns:Ports" minOccurs="1" maxOccurs="1"/>
    </sequence>
    <attribute name="AssociationID" type="string" use="required"/>
</complexType>
```

It is characterized by the attribute *AssociationID* which uniquely identifies it and consist of the following elements:

- **Domains** :

```
<complexType name="Domains">
    <sequence>
        <element name="Domain"
            type="Domain" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

It is composed by a sequence of *Domain*:

```
<complexType name="Domain">
    <attribute name="DomainValue" type="anyURI" use="required"/>
</complexType>
```

Each *Domain* is characterized by the attribute *DomainValue* which identify the URI of the *Domain* to block.

- **URLS** :

```
<complexType name="URLS">
    <sequence>
        <element name="URL"
            type="URL" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

It is composed by a sequence of *URL*:

```
<complexType name="URL">
    <attribute name="URLValue" type="anyURI" use="required"/>
</complexType>
```

Each *URL* is characterized by the attribute *URLValue* which identify the URI of the *URL* to block.

- **Ports** :

```
<complexType name="Ports">
    <sequence>
        <element name="Port"
            type="Port" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

It is composed by a sequence of *Port*:

```
<simpleType name="Port">
    <restriction base="integer">
        <minInclusive value="0"/>
        <maxInclusive value="65535"/>
    </restriction>
</simpleType>
```

Each *Port* is represented by an integer value belonging to a predefined range [19].

5.1.3 TrafficTargets

```
<complexType name="TrafficTargets">
    <sequence>
        <element name="TrafficTarget"
               type="tns:TrafficTarget" minOccurs="1" maxOccurs="unbounded
                  ↪ "/>
    </sequence>
</complexType>
```

The *TrafficTargets* consists of a series of *TrafficTarget*, each of which is characterized by a unique identifier: the *TrafficTargetID*.

The idea is similar to that related to the *BlackList* element: identify groups with a unique name so that they can be used.

This is addressed to experienced users who want to express particular policies related to IP addresses. In fact, each *TrafficTarget* consists of a *Targets* element formed in turn by a series of *Target* where each can be an IPv4 address, IPv6 or their range.

In this way the network administrator can easily block all traffic directed, for example, to a particular company or constrain it to the private network.

TrafficTarget

```
<complexType name="TrafficTarget">
    <sequence>
        <element name="Targets"
            type="Targets" minOccurs="1" maxOccurs="1"/>
    </sequence>
    <attribute name="TrafficTargetID" type="string" use="required"/>
</complexType>
```

It is characterized the attribute *TrafficTargetID* which uniquely identifies it and is composed by the *Targets* element.

Targets

```
<complexType name="Targets"
    minOccurs="1" maxOccurs="1">
    <sequence>
        <element name="Target"
            type="tns:Target" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

The *Targets* element is composed by a sequence of *Target*.

Target

```
<complexType name="Target">
    <choice>
        <element name="IPv4Address"
               type="tns:IPv4Address" minOccurs="0" maxOccurs="unbounded"/
                   ↪ >
        <element name="IPv6Address"
               type="tns:IPv6Address" minOccurs="0" maxOccurs="unbounded"/
                   ↪ >
        <element name="IPv4AddressPool"
               type="tns:IPv4AddressPool" minOccurs="0" maxOccurs="
                   ↪ unbounded"/>
        <element name="IPv6AddressPool"
               type="tns:IPv6AddressPool" minOccurs="0" maxOccurs="
                   ↪ unbounded"/>
    </choice>
</complexType>
```

Each *Target* can be a choice between the following elements:

- *IPv4Address*, represents an IPv4 address. The type is obtained by regular expression;
- *IPv6Address*, represents an IPv6 address. The type is obtained by regular expression;
- *IPv4AddressPool*, represents a pool of IPv4 addresses. It consists of an initial IPv4 address and a final IPv4 address;
- *IPv6Address*, represents a pool of IPv6 addresses. It consists of an initial IPv6 address and a final IPv6 address.

```
<simpleType name="IPv4Address">
    <annotation>
        <documentation>IPv4 address in dot-decimal notation. Equivalent to
            ↪  [0-255].[0-255].[0-255].[0-255]</documentation>
    </annotation>
    <restriction base="string">
        <pattern value="((1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.)
            ↪ {3}(1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])" />
    </restriction>
</simpleType>
```

```
<simpleType name="IPv6Address">
    <annotation>
        <documentation>IPv6 address in dot-hexadecimal notation.
            ↪ Equivalent to [0-ffff].[0-ffff].[0-ffff].[0-ffff]</
            ↪ documentation>
    </annotation>
    <restriction base="string">
        <pattern value="([A-Fa-f0-9]{1,4}:){7}[A-Fa-f0-9]{1,4}" />
    </restriction>
</simpleType>
```

```
<complexType name="IPv4AddressPool">
    <sequence>
        <element name="IPv4Starting"
            type="tns:IPv4Address" minOccurs="1" maxOccurs="1"/>
        <element name="IPv4Ending"
            type="tns:IPv4Address" minOccurs="1" maxOccurs="1"/>
    </sequence>
</complexType>
```

```
<complexType name="IPv6AddressPool">
    <sequence>
        <element name="IPv6Starting"
            type="tns:IPv6Address" minOccurs="1" maxOccurs="1"/>
        <element name="IPv6Ending"
            type="tns:IPv6Address" minOccurs="1" maxOccurs="1"/>
    </sequence>
</complexType>
```

5.1.4 Users

```
<complexType name="Users">
    <sequence>
        <element name="Authors"
               type="tns:Authors" minOccurs="1" maxOccurs="1"/>
        <element name="Subjects"
               type="tns:Subjects" minOccurs="1" maxOccurs="1"/>
    </sequence>
</complexType>
```

The *Users* represent the "people" in the network, could be the *Authors*, who are network administrators and in particular those who write policies, or the *Subjects*, who are the target of policies.

It is important to keep in mind that each author can write policies addressed to different subjects, as well as to different subjects may be directed policies by different authors.

Authors

```
<complexType name="Authors">
    <sequence>
        <element name="Author"
               minOccurs="1" maxOccurs="unbounded">
            <complexType>
                <attribute name="AuthorID" type="string" use="required"/>
            </complexType>
        </element>
    </sequence>
</complexType>
```

It is composed by a sequence of *Author*.

The *Author* represents the person who writes a policy, he is characterized by the attribute *AuthorID* which uniquely identifies him.

In the future it could be characterized by other elements such as the *Degree* and *Rights*.

Subjects

```
<complexType name="Subjects">
    <sequence>
        <element name="SpecificUser"
               type="tns:SpecificUser" minOccurs="1" maxOccurs="1"/>
        <element name="UserGroup"
               type="tns:UserGroup" minOccurs="1" maxOccurs="1"/>
    </sequence>
</complexType>
```

The *Subjects* represent the people to whom policies are directed.

Are divided in:

- **SpecificUser**:

```
<complexType name="SpecificUser">
    <sequence>
        <element name="User"
               type="User" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

The *SpecificUser*s are users (for example: a parent, a child, etc.). The element is composed by a sequence of *User*.

```
<complexType name="User">
    <attribute name="UserID" type="string" use="required"/>
    <attribute name="Username" type="string" use="optional"/>
    <attribute name="AddressIPv4" type="tns:IPv4Address" use="
        ↪ optional"/>
    <attribute name="AddressIPv6" type="tns:IPv6Address" use="
        ↪ optional"/>
</complexType>
```

The *User* is characterized by the attribute *UserID* which uniquely identifies him and optionally by a name (*Username*), an IPv4 or an IPv6 address.

In the future it could be characterized by other elements such as *Degree*, *Rights*, *Priority* and so on.

- **UserGroup**:

```
<complexType name="UserGroup">
    <sequence>
        <element name="Group"
               type="Group" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

They are groups made up of users defined before. The element *UserGroup* is composed by a sequence of *Group*.

```
<complexType name="Group">
    <sequence>
        <element name="UserRef"
            type="UserRef" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="GroupID" type="string" use="required"/>
</complexType>
```

Each *Group* is characterized by the attribute *GroupID* which uniquely identifies it and contains a reference (*UserRef*) for each user that is part of it.

```
<complexType name="UserRef">
    <attribute name="UserID" type="string" use="required"/>
</complexType>
```

The *UserRef* refers to the *User* by the attribute *UserID*.

5.1.5 PolicyRepositoryHPL

```
<complexType name="PolicyRepositoryHPL">
    <sequence>
        <element name="Templates"
               type="tns:Templates" minOccurs="1" maxOccurs="1"/>
        <element name="PoliciesHPL"
               type="tns:PoliciesHPL" minOccurs="1" maxOccurs="1"/>
    </sequence>
</complexType>
```

The *PolicyRepositoryHPL* is the main element, contains the policies expressed in High-level Policy Language.

It is composed by *Templates* and *PoliciesHP*.

5.1.6 Templates

```
<complexType name="Templates">
    <sequence>
    <element name="Template"
           type="tns:Template" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

It consists of a series of *Template*.

Template

```
<complexType name="Template">
    <sequence>
        <element name="Operation"
               type="tns:Operation" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="TemplateID" type="string" use="required"/>
</complexType>
```

The basic structure of politics, as seen in 3.1, in this framework takes the name of Operation, which will be described in detail shortly.

The idea of the *Template* is to merge multiple Operations together so as to simplify the process of writing complicated policies, as each template is characterized by an identifier and therefore you can refer to it immediately.

As an instance, consider the scenario in which the network administrator must write a set of policies for the IDP and IPS prevention software to be activated simultaneously and that an antivirus check is started to scan all files and emails for malware.

Therefore, the administrator must write for each user:

1. a policy to start the IDS;
2. a policy to start the IPS;
3. a policy to start the antivirus;
4. a policy to start file checking;
5. a policy to start email checking;

Doing so the administrator wastes a lot of time to write repetitive policies, time that could take to focus on the goal.

For this need to be resolved, the Template concept has been introduced.

The administrator can enter all these operations in a single Template and then write for each user a single policy in which he says: “enable the Template 1”.

So far we've talked about *Operations*, we've seen that the template can merge more together but what are they in practice?

5.1.7 Operation

```
1  <complexType name="Operation">
2     <sequence>
3        <element name="Action"
4              type="tns:Action" minOccurs="1" maxOccurs="1"/>
5        <element name="Object"
6              type="tns:Object" minOccurs="1" maxOccurs="1"/>
7        <element name="Field"
8              type="tns:Field" minOccurs="0" maxOccurs="unbounded"/>
9     </sequence>
10 </complexType>
```

The structure is similar to that described in 3.1, it is the heart of the High-level Policy Language.

```
Action Object [ Field ... Field ]
```
(4.1)

Therefore, we have the *Action*, the *Object* and an optional list of *Field*.

For example, if we want to block the Internet traffic we can use the expression:

```
not_authorized_access Internet_traffic
```

Where:

- "not_authorized_access" is the *Action*
- "Internet_traffic" is the *Object*

If we want to block internet traffic towards social networks:

```
not_authorized_access Internet_traffic social_network
```

Where:

- "not_authorized_access" is the *Action*
- "Internet_traffic" is the *Object*
- "social_network" is the *Field*, in particular it is the *ContentType*

If we want to block internet traffic towards social networks in a certain time:

```
not_authorized_access Internet_traffic social_network
            Monday from 2 p.m. to 7 p.m.
```

Where:

- "not_authorized_access" is the *Action*
- "Internet_traffic" is the *Object*
- "social_network" is the *Field*, in particular it is the *ContentType*
- "Monday from 2 p.m. to 7 p.m." is the *Field*, in particular it is the *Time*

Now we analyze the previous elements in detail, but it is important to take in consideration that not all the elements are compatible. There are some restrictions to consider as the Objects cannot be associated with all the Actions and nor with all the Fields.

The full list of Object compatible Actions is shown in the Fig. 4.4 while the other in the Fig. 4.4.

Object **Action**	is/are not authorized to access	is/are authorized to access	enable(s)	remove(s)	reduce(s)	check(s) over	count(s)	protect(s) conf.	protect(s) integr.	protect(s) conf.integr.
VoIP traffic	+	+						+	+	+
P2P traffic	+	+						+	+	+
3G/4G traffic	+	+						+	+	+
Internet traffic	+	+						+	+	+
intranet traffic	+	+						+	+	+
DNS traffic	+	+					+	+	+	+
all traffic	+	+						+	+	+
public identity				+						
Resource 'X'	+	+								
file scanning			+							
email scanning			+							
antivirus			+							
basic parental control			+							
advance parental control			+							
IDS/IPS			+							
DDos attack protection			+							
tracking techniques				+						
advertisement				+						
bandwidth					+					
security status						+				
connection							+			
logging			+							

Figure 5.1: The object compatible actions

Object / Field	time period	traffic target	specific URL	type of content	purpose	bandwidth value	resource value
VoIP traffic	+	+					
P2P traffic	+	+					
3G/4G traffic	+	+					
Internet traffic	+	+	+	+			
intranet traffic	+	+	+	+			
DNS traffic	+	+	+	+			
all traffic	+	+	+	+			
public identity	+	+	+				
Resource							+
file scanning					+		
email scanning					+		
antivirus		+					
basic parental control							
advance parental control	+			+			
IDS/IPS		+					
DDos attack protection		+					
tracking techniques			+				
advertisement			+				
bandwidth	+		+			+	
security status							
connection	+		+				
logging	+	+	+	+			

Figure 5.2: The object compatible fields

5.1.8 Action

```
<simpleType name="Action">
    <restriction base="string">
        <enumeration value="authorized_access"/>
        <enumeration value="not_authorized_access"/>
        <enumeration value="enable"/>
        <enumeration value="remove"/>
        <enumeration value="reduce"/>
        <enumeration value="check_over"/>
        <enumeration value="count"/>
        <enumeration value="protect_confidentiality"/>
        <enumeration value="protect_integrity"/>
    </restriction>
</simpleType>
```

It is the deed (e.g. enable, protect, authorized) that the administrator wants must be performed on the object.

5.1.9 Object

```
<simpleType name="Object">
   <restriction base="string">
      <enumeration value="Internet_traffic"/>
      <enumeration value="P2P_traffic"/>
      <enumeration value="DNS_traffic"/>
      <enumeration value="DNS_request"/>
      <enumeration value="DNS_response"/>
      <enumeration value="Intranet_traffic"/>
      <enumeration value="VoIP_traffic"/>
      <enumeration value="traffic_3G"/>
      <enumeration value="traffic_4G"/>
      <enumeration value="all_traffic"/>
      <enumeration value="public_identity"/>
      <enumeration value="resource"/>
      <enumeration value="basic_parental_control"/>
      <enumeration value="advanced_parental_control"/>
      <enumeration value="antivirus"/>
      <enumeration value="logging"/>
      <enumeration value="IDS"/>
      <enumeration value="IPS"/>
      <enumeration value="DDos_attack_protection"/>
      <enumeration value="email_scanning"/>
      <enumeration value="file_scanning"/>
      <enumeration value="tracking_techniques"/>
      <enumeration value="advertisement"/>
      <enumeration value="bandwidth"/>
      <enumeration value="connection"/>
      <enumeration value="security_status"/>
   </restriction>
</simpleType>
```

It is the resource (e.g. Internet, antivirus, DNS, IDS) that is the target of the Action.

5.1.10 Field

```
<complexType name="Field">
    <choice>
        <element name="Time"
               type="tns:Time" minOccurs="0" maxOccurs="1"/>
        <element name="SpecificURL"
               type="tns:ListURL" minOccurs="0" maxOccurs="1"/>
        <element name="ContentType"
               type="tns:ContentType" minOccurs="0" maxOccurs="1"/>
        <element name="TrafficTarget"
               type="string" minOccurs="0" maxOccurs="1"/>
        <element name="Purpose"
               type="tns:Purpose" minOccurs="0" maxOccurs="1"/>
        <element name="Bandwidth"
               type="tns:Bandwidth" minOccurs="0" maxOccurs="1"/>
    </choice>
</complexType>
```

It is an optional condition (e.g. time, traffic target, content type) that allow to better specify the action on the object. It can be chosen from the following element.

Time

```
<complexType name="Time">
    <sequence>
        <element name="TimeInterval"
               type="tns:TimeInterval" minOccurs="1" maxOccurs="unbounded"
                  ↪ />
    </sequence>
</complexType>
```

The *Time* element is used to express conditions linked to the passage of time. It consists of a sequence of intervals.

TimeInterval:

```
<complexType name="TimeInterval">
    <sequence>
        <element name="DayOfWeek"
            type="tns:Day" minOccurs="0" maxOccurs="1"/>
        <element name="TimePeriod"
            type="tns:TimePeriod" minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="timezone" type="string" use="required"/>
</complexType>
```

Each *TimeInterval* is characterized by an information about the reference timezone and consists of a sequence of days (*DayOfWeek*) and time slots (*TimePeriod*).

DayOfWeek:

```
<simpleType name="Day">
   <restriction base="string">
      <enumeration value="Monday"/>
      <enumeration value="Tuesday"/>
      <enumeration value="Wednesday"/>
      <enumeration value="Thursday"/>
      <enumeration value="Friday"/>
      <enumeration value="Saturday"/>
      <enumeration value="Sunday"/>
      <enumeration value="Weekend"/>
   </restriction>
</simpleType>
```

The *DayOfWeek* is the day and it can be a day of the week or the keyword “Weekend” identifying a particular set of days.

TimePeriod:

```
<complexType name="TimePeriod">
   <sequence>
      <element name="StartTime" type="tns:myTime"/>
      <element name="EndTime" type="tns:myTime"/>
   </sequence>
</complexType>
```

The *TimePeriod* element is used to represent a specific time frame, which is characterized by a beginning and an end. Both are expressed with a particular type.

myTime:

```
<complexType name="myTime">
    <attribute name="date" type="tns:regExrDate" use="optional"/>
    <attribute name="time" type="tns:regExrTime" use="required"/>
</complexType>

<simpleType name="regExrDate">
    <restriction base="string">
        <pattern value="
            (\d{4})
            (\.|:|\\)
            (([0]{0,1}[1-9])|(1{1}[0-2]{1}))
            (\.|:|\\)
            (([0-2]){0,1}[0-9]{1}|(3{1}[0-1]{1}))
            "/>
    </restriction>
</simpleType>
```

It is the type of data used to specify the time interval, it uses a particular regular expression.

ContentType

```
<complexType name="ContentType">
    <sequence>
        <element name="SpecificContent"
                minOccurs="1" maxOccurs="unbounded">
            <complexType>
                <attribute name="associationID" type="string" use="required
                    ↪ "/>
            </complexType>
        </element>
    </sequence>
</complexType>
```

The Field *ContentType* is used to specify one or more specific content. Each characterized by the *associationID*, which refers to the *Association* defined above. Indeed, in the *BlackList* (5.1.2), belonging to the *KnowledgeBase*, all the *Associations* are declared so that they can be used here.

As an example, the network administrator can block the Internet traffic directed to "Social Networks" by simply specifying it with the Field *ContentType* as it has previously created an *Association* that links the word "Social Network" to all urls or domains to block.

SpecificURL

```
<complexType name="ListURL">
    <sequence>
        <element name="URL"
        minOccurs="0" maxOccurs="unbounded">
            <complexType>
                <attribute name="URLValue" type="anyURI" use="required"/>
            </complexType>
        </element>
    </sequence>
</complexType>
```

The Field *SpecificURL* is used if we want to specify one or more URL addresses. For example, when we want to block the Internet traffic to a particular address.

TrafficTarget

The Field *TrafficTarget* is a string, which refers to the traffic target described in (5.1.3). In this way it is possible to operate on the previously defined IP addresses, simply acting on this Field.

Bandwidth

```
<complexType name="Bandwidth">
    <attribute name="Value" type="integer" use="required"/>
    <attribute name="Unit" type="tns:Bandwidth_Unit" use="required"/>
</complexType>

<simpleType name="Bandwidth_Unit">
    <restriction base="string">
        <enumeration value="Kbit/s"/>
        <enumeration value="Mbit/s"/>
        <enumeration value="Gbit/s"/>
    </restriction>
</simpleType>
```

The Field *Bandwidth* is used when we want specify the bandwidth to reduce.

Purpose

```
<complexType name="Purpose">
    <sequence>
        <element name="PurposeDetection"
            minOccurs="1" maxOccurs="unbounded">
            <complexType>
                <attribute name="detection" type="tns:Detection" use="
                    ↪ required"/>
            </complexType>
        </element>
    </sequence>
</complexType>

<simpleType name="Detection">
    <restriction base="string">
        <enumeration value="adware"/>
        <enumeration value="backdoor"/>
        <enumeration value="botnet"/>
        <enumeration value="dialers"/>
        <enumeration value="exploit"/>
        <enumeration value="keylogger"/>
        <enumeration value="hijackers "/>
        <enumeration value="malware"/>
        <enumeration value="ransomware"/>
        <enumeration value="rogueware"/>
        <enumeration value="rootkit"/>
        <enumeration value="spam"/>
        <enumeration value="spyware"/>
        <enumeration value="trojan"/>
        <enumeration value="virus"/>
        <enumeration value="worm"/>
    </restriction>
</simpleType>
```

The Field *Purpose* is used if we want to specify one or more target to detect.

For example, we can run a scan with the purpose of detecting in particular “malware”.

5.1.11 PoliciesHP

```
<complexType name="PoliciesHPL">
    <sequence>
        <element name="PolicyHPL"
              type="tns:PolicyHPL" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

It consists of a series of *PolicyHPL*:

PolicyHPL

```
<complexType name="PolicyHPL">
    <sequence>
        <element name="Author"
               type="string" minOccurs="1" maxOccurs="1"/>
        <element name="PolicyRuleHPL"
               type="tns:PolicyRuleHPL" minOccurs="1" maxOccurs="unbounded
                  ↪ "/>
    </sequence>
    <attribute name="PolicyID" type="string" use="required"/>
</complexType>
```

The *PolicyHPL* is the key concept of the repository. It is characterized by:

- The attribute *PolicyID* which uniquely identifies it in the whole repository;
- The element *Author* which represents the person who writes the policy. It is a string that makes reference to a user which is an "Author" defined before in (5.1.4).
- The list of all the *PolicyRuleHPL* wrote by this *Author*.

5.1.12 PolicyRuleHPL

```
<complexType name="PolicyRuleHPL">
    <sequence>
        <element name="Subject"
               type="tns:Subject" minOccurs="1" maxOccurs="1"/>
        <choice>
            <element name="Operation"
                   type="tns:Operation" minOccurs="0" maxOccurs="1"/>
            <element name="Template"
                   type="string" minOccurs="0" maxOccurs="1"/>
        </choice>
    </sequence>
</complexType>
```

The *PolicyRuleHPL* represents one of the *Rules* wrote by an *Author*. It is addressed to a *Subject*, who, as seen in 5.1.4, can be a specific user or a group of users.

The *PolicyRuleHPL* can be:

- an *Operation*, as described in 5.1.7, in this way the author can address to the *Subject* a precise *Action*;
- a *Template*, as described in 5.1.6, in order to specify a set of *Actions* for this *Subject*;

5.2 NSF Catalog

The *NSF Catalog* represents the catalog of the network functions which are available in the system. This catalog was created with the aim of giving administrators a way to easily list in a complete and detailed way all the network functions available in the system. The catalogue has been designed to contain all the information related to the functions, so as to have a full awareness when you then go to choose them.

It is presented in this way:

```
<element name="NSFCatalog">
    <sequence>
        <element name="Capabilities"
               type="tns:Capabilities" minOccurs="1" maxOccurs="1"/>
        <element name="NSFs"
               type="tns:NSFs" minOccurs="0" maxOccurs="1"/>
    </sequence>
</element>
```

It is composed by the *NSFs*, which represents the real catalogue, it contains all the functions *NSF* that make up the catalog of the system.

```
<complexType name="NSFs">
    <sequence>
        <element name="NSF"
               type="tns:NSF" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

5.2.1 NSF

```
<complexType name="NSF">
    <sequence>
        <element name="GeneralInfo"
               type="tns:GeneralInfo" minOccurs="1" maxOccurs="1"/>
        <element name="SoftwareInfo"
               type="tns:SoftwareInfo" minOccurs="1" maxOccurs="1"/>
        <element name="HardwareInfo"
               type="tns:HardwareInfo" minOccurs="1" maxOccurs="1"/>
        <element name="Functionality"
               type="tns:Functionality" minOccurs="1" maxOccurs="1"/>
    </sequence>
    <attribute name="NSF_ID"
           type="string"
           use="required"/>
</complexType>
```

Each function *NSF* that belongs to the catalog is characterized by the attribute *NSF_ID* that uniquely identifies it. It contains a range of information of various kinds that allow you to make an informed choice at the moment when you go to choose it.

This is because the choice of functions could be dictated by particular conditions. For example, you want to choose functions so as to reduce the overall cost or minimizing the amount of resources (e.g. ram, disk, cpu, etc.) needed.

The most important motivation that leads a function to be chosen is its ability to satisfy a certain policy. Therefore, each function must contain within it a clear list of what it is in case to support.

Therefore, it contains inside it:

- *GeneralInfo*: all the information about the vendor and the website;
- *SoftwareInfo*: all the information concerning the software part as author, developers, release and version, repository, licence, Operating Systens supported and programming languages used in the code;
- *HardwareInfo*: all the information concerning the resources hardware as cpu, ram and disk but also information regarding the bandwidth and delay;
- *Functionality*: it is the key information, it contains the knowledge about the "capabilities" that are supported by the NSF (Network Security Function). The capability represents the intrinsic feature of the network function, that is what that function knows and can do.

5.2.2 GeneralInfo

```
1 <complexType name="GeneralInfo">
2    <sequence>
3       <element name="Vendor" type="string" minOccurs="0" maxOccurs="1"/>
4       <element name="WebSite" type="anyURI" minOccurs="0" maxOccurs="1"/
            ↪ >
5    </sequence>
6 </complexType>
```

The *GeneralInfo* represents the information about the vendor and the website. It is composed by the following elements:

- *Vendor*, it is a string representing the name of the seller;
- *WebSite*, it is the URI where it is possible to find additional information about the NSF.

5.2.3 SoftwareInfo

```
<complexType name="SoftwareInfo">
    <sequence>
        <element name="Author"
            type="string" minOccurs="0" maxOccurs="1"/>
        <element name="Developers"
            type="Developers" minOccurs="0" maxOccurs="1"/>
        <element name="VersionInfo"
            type="VersionInfo"minOccurs="0" maxOccurs="1"/>
        <element name="Repository"
            type="anyURI" minOccurs="0" maxOccurs="1"/>
        <element name="ProgrammingLanguagesUsed"
            type="ProgrammingLanguagesUsed" minOccurs="0" maxOccurs="1"/>
        <element name="OperatingSystemsSupported"
            type="OSSupported" minOccurs="0" maxOccurs="1"/>
        <element name="Licence"
            type="tns:Licence" minOccurs="0" maxOccurs="1"/>
    </sequence>
</complexType>
```

The *SoftwareInfo* element contains information about the software. It is composed by the following elements:

- *Author*, he is the author and so the owner of the software;
- *Developers*, they are those who have helped to develop the software;
- *VersionInfo*, information about the version;
- *Repository*, that is the link to the repository of the code;
- *ProgrammingLanguagesUsed*, all the programming languages used in the development;
- *OperatingSystemsSupported*, all the Operating Systems supported by the NSF;
- *Licence*, the licence of publication;

Author

The *Author* is represented by the *string* with his name.

Developers

```
<complexType name="Developers">
    <sequence>
        <element name="Developer"
            type="string" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

They are those who have helped to develop the software, can be one or more.

VersionInfo

```
<complexType name="VersionInfo">
    <attribute name="version" type="tns:regExrVersion" use="required"/>
    <attribute name="release" type="tns:regExrVersion" use="required"/>
    <attribute name="releaseDate" type="tns:regExrDate" use="required"/>
</complexType>

<simpleType name="regExrVersion">
    <restriction base="string">
        <pattern value="((\d+)(\.\d+)*)"/>
    </restriction>
</simpleType>

<simpleType name="regExrDate">
    <restriction base="string">
        <pattern value="(\d{4})(\.|:|\\)(([0]{0,1}[1-9])|(1{1}[0-2]{1}))
            ↪ (\.|:|\\)(([0-2]){0,1}[0-9]{1}|(3{1}[0-1]{1}))"/>
    </restriction>
</simpleType>
```

The information about the version are contained in the following attributes:

- *version*, the number of the version;
- *release*, the number of the release;
- *releaseDate*, the date on which the release was issued;

OperatingSystemsSupported

```
<complexType name="OSSupported">
    <attribute name="OS_Type"   type="tns:OS_Type"/>
    <attribute name="OS_Version"  type="string"/>
    <attribute name="OS_Architecture" type="tns:Architecture"/>
</complexType>

<simpleType name="OS_Type">
    <restriction base="string">
        <enumeration value="ARM"/>
        <enumeration value="Android"/>
        <enumeration value="BSD"/>
        <enumeration value="CentOS"/>
        <enumeration value="Debian"/>
        <enumeration value="Fedora"/>
        <enumeration value="Fink"/>
        <enumeration value="FreeBSD"/>
        <enumeration value="OpenBSD"/>
        <enumeration value="Gentoo"/>
        <enumeration value="IBM"/>
        <enumeration value="iOS"/>
        <enumeration value="Linux"/>
        <enumeration value="macOS"/>
        <enumeration value="Maemo"/>
        <enumeration value="Mandriva"/>
        <enumeration value="NetBSD"/>
        <enumeration value="Slackware"/>
        <enumeration value="SLES"/>
        <enumeration value="Solaris"/>
        <enumeration value="Ubuntu"/>
        <enumeration value="Unix"/>
        <enumeration value="Windows"/>
        <enumeration value="Other"/>
        <enumeration value="Unspecified"/>
    </restriction>
</simpleType>

<simpleType name="Architecture">
    <restriction base="string">
        <enumeration value="x86"/>
        <enumeration value="x64"/>
        <enumeration value="ARM"/>
        <enumeration value="Unspecified"/>
    </restriction>
</simpleType>
```

The information about the operating systems supported are expressed by the following attributes:

- *OS_Type*, the enumerative rapresenting the ownership;
- *OS_Version*, the version of the OS;
- *OS_Architecture*, the architecture of the current version of the OS;

ProgrammingLanguagesUsed

```
<complexType name="ProgrammingLanguagesUsed">
<sequence>
    <element name="ProgrammingLanguage"
        type="tns:ProgrammingLanguage" minOccurs="1" maxOccurs="unbounded"
            ↪ />
</sequence>
</complexType>

<simpleType name="ProgrammingLanguage">
    <restriction base="string">
        <enumeration value="Assembly"/>
        <enumeration value="C"/>
        <enumeration value="C++"/>
        <enumeration value="C#"/>
        <enumeration value="Delphi"/>
        <enumeration value="Java"/>
        <enumeration value="JavaScript"/>
        <enumeration value="Matlab"/>
        <enumeration value="Perl"/>
        <enumeration value="Python"/>
        <enumeration value="R"/>
        <enumeration value="Ruby"/>
        <enumeration value="SQL"/>
        <enumeration value="Visual Basic"/>
        <enumeration value="Other"/>
        <enumeration value="Unspecified"/>
    </restriction>
</simpleType>
```

The information about the programming languages used are expressed by enumeratives.

5.2.4 HardwareInfo

```
<complexType name="HardwareInfo">
    <sequence>
        <element name="CPU"
            type="tns:CPU" minOccurs="0" maxOccurs="1"/>
        <element name="RAM"
            type="tns:RAM" minOccurs="0" maxOccurs="1"/>
        <element name="Disk"
            type="tns:Disk" minOccurs="0" maxOccurs="1"/>
        <element name="Bandwidth"
            type="tns:Bandwidth" minOccurs="0" maxOccurs="1"/>
        <element name="Cost"
            type="integer" minOccurs="0" maxOccurs="1"/>
        <element name="Delay"
            type="tns:Delay" minOccurs="0" maxOccurs="1"/>
    </sequence>
</complexType>
```

The *HardwareInfo* element contains information about the hardware. It is composed by the following elements:

- *CPU*, represents the CPU used by the NSF;
- *RAM*, represents the RAM required by the NSF;
- *Disk*, represents the Disk required by the NSF;
- *Bandwidth*, represents the max bandwidth used by the NSF;
- *Cost*, represents the Cost of the NSF;;
- *Delay*, represents the max delay of the NSF.

CPU

```
<complexType name="CPU">
    <attribute name="value" type="decimal"   use="required"/>
    <attribute name="unit" type="tns:CPU_Unit" use="required"/>
</complexType>

<simpleType name="CPU_Unit">
    <restriction base="string">
        <enumeration value="Ghz"/>
        <enumeration value="Mhz"/>
    </restriction>
</simpleType>
```

The *CPU* is represented by the following attributes:

- **value**, it is a *decimal*;
- **unit**, it is a *CPU_Unit* which can be *Ghz* or *Mhz*.

Cost

The *Cost* of the NSF is represented by the *integer*, it is the market price.

RAM

```
1  <complexType name="RAM">
2      <attribute name="value" type="integer"   use="required"/>
3      <attribute name="unit" type="tns:RAM_Unit" use="required"/>
4  </complexType>
5
6  <simpleType name="RAM_Unit">
7      <restriction base="string">
8          <enumeration value="KB"/>
9          <enumeration value="MB"/>
10         <enumeration value="GB"/>
11     </restriction>
12 </simpleType>
```

The *RAM* is represented by the following attributes:

- **value**, it is an *integer*;
- **unit**, it is the *enumerative RAM_Unit* which can be *KB*, *MB* or *GB*.

Disk

```
1  <complexType name="Disk">
2      <attribute name="value" type="integer"   use="required"/>
3      <attribute name="unit" type="tns:Disk_Unit" use="required"/>
4  </complexType>
5
6  <simpleType name="Disk_Unit">
7      <restriction base="string">
8          <enumeration value="KB"/>
9          <enumeration value="MB"/>
10         <enumeration value="GB"/>
11         <enumeration value="TB"/>
12     </restriction>
13 </simpleType>
```

The *Disk* is represented by the following attributes:

- **value**, it is an *integer*;
- **unit**, it is the *enumerative Disk_Unit* which can be *KB*, *MB*, *GB* or *TB*.

Bandwidth

```
<complexType name="Bandwidth">
    <attribute name="value" type="integer" use="required"/>
    <attribute name="unit" type="tns:Bandwidth_Unit" use="required"/>
</complexType>

<simpleType name="Bandwidth_Unit">
    <restriction base="string">
        <enumeration value="Kb/s"/>
        <enumeration value="Mb/s"/>
        <enumeration value="Gb/s"/>
    </restriction>
</simpleType>
```

The *Bandwidth* is represented by the following attributes:

- **value**, it is an *integer*;
- **unit**, it is the *enumerative Bandwidth_Unit* which can be *Kb/s*, *Mb/s* or *Gb/s*.

Delay

```
<complexType name="Delay">
    <attribute name="value" type="integer" use="required"/>
    <attribute name="unit" type="tns:MaxDelay_Unit" use="required"/>
</complexType>

<simpleType name="MaxDelay_Unit">
    <restriction base="string">
        <enumeration value="ms"/>
    </restriction>
</simpleType>
```

The *Delay* is represented by the following attributes:

- **value**, it is an *integer*;
- **unit**, it is the *enumerative Delay_Unit* which can be *ms*.

5.2.5 Functionality

```
<complexType name="Functionality">
    <sequence>
        <element name="CapabilitiesRef"
            type="CapabilitiesRef" minOccurs="1" maxOccurs="1"/>
    </sequence>
</complexType>
```

The *Functionality* element contains information about *Capabilities* supported by the NSF. It is composed by the element *CapabilitiesRef*.

CapabilitiesRef

```
<complexType name="CapabilitiesRef">
    <sequence>
        <element name="CapabilityRef" minOccurs="1" maxOccurs="unbounded">
            <complexType>
                <attribute name="CapabilityID" type="tns:Capability" use="
                    ↪ required"/>
            </complexType>
        </element>
    </sequence>
</complexType>
```

The *CapabilitiesRef* is composed by a sequence of *CapabilityRef*, each of which is characterized by the attribute *CapabilityID* which is an enumerative that represents the identifier of the capability.

Capability

The enumerative *Capability* may currently be one of the following:

```
<simpleType name="Capability">
    <restriction base="string">
        <enumeration value="Packet_Filter_L4_Stateless"/>
        <enumeration value="Packet_Filter_L4_Statefull"/>
        <enumeration value="FW_Application"/>
        <enumeration value="FW_Web_Application"/>
        <enumeration value="Time_Filter"/>
        <enumeration value="Proxy_Forward"/>
        <enumeration value="Proxy_Reverse"/>
        <enumeration value="ProtectConfidentiality"/>
        <enumeration value="ProtectIntegrity"/>
        <enumeration value="Parental_Control"/>
        <enumeration value="Logging"/>
        <enumeration value="IDS"/>
        <enumeration value="IPS"/>
        <enumeration value="Antivirus"/>
        <enumeration value="FileScanning"/>
        <enumeration value="EmailScanning"/>
        <enumeration value="WebScanning"/>
        <enumeration value="DDosAttackProtection"/>
        <enumeration value="BlockAdvertisement"/>
        <enumeration value="BlockTracking"/>
        <enumeration value="IdentityProtection"/>
        <enumeration value="BandwidthManagement"/>
    </restriction>
</simpleType>
```

Their origin is explained in the next section.

5.3 Capabilities

The *Capabilities* collects all the *Capability* which constitutes the key concept underlying the selection of functions;

```
<complexType name="Capabilities">
    <sequence>
        <element name="Capability"
            type="tns:Capability" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

The *Capabilities* element represents a collection of **Capability**.

```
<complexType name="Capability">
    <sequence>
        <element name="Features"
            type="tns:Features" minOccurs="1" maxOccurs="1"/>
        </sequence>
            <attribute name="CapabilityID" type="string" use="required"/>
</complexType>
```

Each *Capability* is characterized by an identifier (*CapabilityID*) that uniquely identifies. It represents the implicit characteristic of a class of functions, as "Packet_Filter_L4_Stateless".

It consists of a series of features as described in (4.2.2).

```
<complexType name="Features">
    <sequence>
        <element name="Feature"
            type="string" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

Each *Feature* is a string which represents a detail of the capability. As an instance, the *Capability* "Packet_Filter_L4_Stateless" contains the *"Packet_Filter"* as *Feature*.

The full list of currently supported capabilities is described below:

```
<Capabilities>
    <Capability CapabilityID=‘‘Packet_Filter_L4_Stateless’’>
        <Features>
            <Feature>Packet_Filter</Feature>
            <Feature>L4</Feature>
            <Feature>Stateless</Feature>
        </Features>
    </Capability>

    <Capability CapabilityID=‘‘Packet_Filter_L4_Statefull’’>
        <Features>
            <Feature>Packet_Filter</Feature>
            <Feature>L4</Feature>
            <Feature>Statefull</Feature>
        </Features>
    </Capability>

    <Capability CapabilityID=‘‘FW_Application’’>
        <Features>
            <Feature>Firewall</Feature>
            <Feature>L7</Feature>
        </Features>
    </Capability>

    <Capability CapabilityID=‘‘FW_Web_Application’’>
        <Features>
            <Feature>Firewall</Feature>
            <Feature>HTTP_traffic</Feature>
        </Features>
    </Capability>

    <Capability CapabilityID=‘‘Time_Filter’’>
        <Features>
            <Feature>Time</Feature>
        </Features>
    </Capability>

    <Capability CapabilityID=‘‘Proxy_Forward’’>
        <Features>
            <Feature>Proxy</Feature>
            <Feature>Forward</Feature>
        </Features>
    </Capability>
```

```
49
50      <Capability CapabilityID=‘‘Proxy_Reverse’’>
51          <Features>
52              <Feature>Proxy</Feature>
53              <Feature>Reverse</Feature>
54          </Features>
55      </Capability>
56
57      <Capability CapabilityID=‘‘ProtectConfidentiality’’>
58          <Features>
59              <Feature>Protect</Feature>
60              <Feature>Confidentiality</Feature>
61          </Features>
62      </Capability>
63
64      <Capability CapabilityID=‘‘ProtectIntegrity’’>
65          <Features>
66              <Feature>Protect</Feature>
67              <Feature>Integrity</Feature>
68          </Features>
69      </Capability>
70
71      <Capability CapabilityID=‘‘Parental_Control’’>
72          <Features>
73              <Feature>Parental_Control</Feature>
74          </Features>
75      </Capability>
76
77      <Capability CapabilityID=‘‘Logging’’>
78          <Features>
79              <Feature>Logging</Feature>
80          </Features>
81      </Capability>
82
83      <Capability CapabilityID=‘‘IDS’’>
84          <Features>
85              <Feature>IDS</Feature>
86          </Features>
87      </Capability>
88
89      <Capability CapabilityID=‘‘IPS’’>
90          <Features>
91              <Feature>IPS</Feature>
92          </Features>
93      </Capability>
94
95
96
97
98
```

```
99
100     <Capability CapabilityID=‘‘Antivirus’’>
101         <Features>
102             <Feature>Antivirus</Feature>
103         </Features>
104     </Capability>
105
106     <Capability CapabilityID=‘‘FileScanning’’>
107         <Features>
108             <Feature>File</Feature>
109             <Feature>Scanning</Feature>
110         </Features>
111     </Capability>
112
113     <Capability CapabilityID=‘‘EmailScanning’’>
114         <Features>
115             <Feature>Email</Feature>
116             <Feature>Scanning</Feature>
117         </Features>
118     </Capability>
119
120     <Capability CapabilityID=‘‘WebScanning’’>
121         <Features>
122             <Feature>Web</Feature>
123             <Feature>Scanning</Feature>
124         </Features>
125     </Capability>
126
127     <Capability CapabilityID=‘‘DDosAttackProtection’’>
128         <Features>
129             <Feature>Attack</Feature>
130             <Feature>DDos</Feature>
131             <Feature>Protection</Feature>
132         </Features>
133     </Capability>
134
135     <Capability CapabilityID=‘‘BlockAdvertisement’’>
136         <Features>
137             <Feature>Block</Feature>
138             <Feature>Advertisement</Feature>
139         </Features>
140     </Capability>
141
142     <Capability CapabilityID=‘‘BlockTracking’’>
143         <Features>
144             <Feature>Block</Feature>
145             <Feature>Tracking</Feature>
146         </Features>
147     </Capability>
148
```

```
149
150     <Capability CapabilityID=‘‘IdentityProtection’’>
151         <Features>
152             <Feature>Identity</Feature>
153             <Feature>Protection</Feature>
154         </Features>
155     </Capability>
156
157     <Capability CapabilityID=‘‘BandwidthManagement’’>
158         <Features>
159             <Feature>Bandwidth</Feature>
160             <Feature>Management</Feature>
161         </Features>
162     </Capability>
163 </Capabilities>
```

5.4 Hosts

In this session will be shown the *Hosts* element, it represents the physical server where it is possible to allocate the instances of functions.

```
<element name="Hosts">
    <complexType >
        <sequence>
            <element ref="tns:Host"
                minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
</element>
```

It is composed by a sequence of Host.

Host

```
<element name="Host">
    <complexType>
        <sequence>
            <element name="SupportedVNF"
            type="string" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="HostID"  type="string" use="required"/>
        <attribute name="cpu"   type="integer" use="required"/>
        <attribute name="ram"   type="integer" use="required"/>
        <attribute name="disk"   type="integer" use="required"/>
        <attribute name="bandwidth" type="integer" use="required"/>
    </complexType>
</element>
```

Each *Host* is characterized by the following attributes:

- *HostID*, it is the identifier that uniquely identifies it;
- *cpu*, is the maximum value of CPU available and usable by the functions;
- *ram*, is the maximum value of RAM available and usable by the functions;
- *disk*, is the maximum value of Disk available and usable by the functions;
- *bandwidth*, is the maximum value of Bandwidth available and usable by the functions;

Additionally, zero or more *SupportedVNF* elements may be present. They are *string* elements which represent which network functions (e.g. Squid, Iptable, etc.) are supported by the host.

Chapter 6

Selection and Optimization phase

This chapter will describe the constants and formulas used by Gurobi to perform the selection, allocation and optimization step within the framework, performed by the SAP module. Will also be presented those that were used by z3 so as to make a comparison.

6.1 Constraints and Objectives

As discussed in previous chapters, the *SAP* (Selection And Placement) module carry out the following tasks:

1. Select how many and which instances of network functions are needed to satisfy the capability instances received as input;
2. Allocate selected instances among the physical hosts available;
3. Optimize these choices based on the information contained in the configuration file.

These tasks are performed using the Gurobi tool. Gurobi allows to automate these tasks associating binary variables, with value 1 (in case of choice) and 0 otherwise, to the various elements (NSF, capability and Host) and relationships between them. The choice of values, assigned to variables, depends on some constraints expressed by mathematical formulas that Gurobi tries to solve.

In order to optimize these values one or more objectives are expressed. Also these objectives (for example minimizing the cost of functions) are expressed by mathematical models that Gurobi tries to solve.

6.1.1 Constraints

The constraints to be met during the assignment of the variables are as follows:

1. Each instance of the NSF is used and must be allocated if and only if there at least one instance of Capability that use it;

2. Each instance of the capabilities must be assigned only once. Therefore, each of it can be allocated in one and one instance of NSF;
3. All the instances of the same capability must be allocated in different instances of function. Therefore, in each instance of a function is possible to allocate only zero or one instance of the same capability. It should be noted that a function instance can allocate multiple instances of capability as long as the capabilities are different.

   As an instance, if we have the scenario composed by the following elements:

   - C1_1, C1_2, C2_1, which are two instances of the capability C1 and one instance of the capability C2;
   - NSF1_1, which are one instance of the NSF1.

   Only one between the two capabilities C1_1 and C1_2 may be associated with NSF1_1 but C2_1 can be associated without problems.
4. Each function instance is allocated in a host only if used;
5. Each instance of function, if used, must be allocated in only an host;
6. Each host is used if and only if there is at least one function which is allocated in it;
7. The sum of all the resources required by the functions to be allocated must be less or equal than the maximum supported by the host;
8. If the variable "VNFmustBeSupportedByHost" in the configuration file is true, the instance of an NSF can be allocated in the host if and only if the host supports it;

All these constraints must be respected in the final solution. In case at least one is not complied, then the whole solution is unsatisfiable.

6.1.2 Multiple-Objectives

The objectives that Gurobi tries to achieve are the following:

- Minimize the *cost* of the functions used;
- Minimize the *cpu* of the functions used;
- Minimize the *ram* of the functions used;
- Minimize the *disk* of the functions used;
- Minimize the *bandwidth* of the functions used;
- Minimize the *number of the functions* used;
- Minimize the *number of the hosts* used;

It is important to note that each of these objectives can be associated with a priority. Gurobi manages the multi-objectives and then will try to solve all the objectives hierarchically starting from the one with the highest priority

6.2 Z3 formulation

As mentioned above, initially the selection and optimization part was carried out through the use of the z3 tool. In this section the formulas and notations used by z3 will be shown.

6.2.1 Z3 Symbols

Table 6.1 defines all symbols constituting the elements and relationships used by z3.

Symbols	Notations
C	Set of capabilities
$c_{i,j}$	The j-th instance of the i-th capability $\in C$
$x^c_{i,j}$	Boolean variable, true if the instance $c_{i,j}$ is requested
N	Set of NSFs
$n_{l,m}$	The m-th instance of the l-th NSF $\in N$
$x^n_{l,m}$	Boolean variable, true if the instance $n_{l,m}$ is used
$y^{l,m}_{i,j}$	Boolean variable, true if the $c_{i,j}$ is supported and assigned to $n_{l,m}$
r_k	String variable, represents the resource; $r_k = \begin{cases} cpu & \text{if } k = 0 \\ disk & \text{if } k = 1 \\ ram & \text{if } k = 2 \\ bandwidth & \text{if } k = 3 \end{cases}$
$w_{k,l}$	Integer variable, it is the weight associated with the resource r_k of the function n_l
H	Set of hosts
w^n_l	Integer variable, it is the weight (cost) associated with the function n_l
h_s	The s-th host $\in H$
x^h_s	Boolean variable, true if the h_s host is used
N^h_s	Set of NSFs supported by the h_s
$z^s_{l,m}$	Boolean variable, true if the instance $n_{l,m}$ is allocated in h_s
W^s_k	Integer variable, it is the weight associated with the resource r_k of the host h_s
$W^l_{a,b}$	Integer variable, it is the weight associated with the link $l_{a,b}$
bool_to_int(α)	Function, converts the α value from boolean to integer

Table 6.1: The symbols used by Z3

6.2.2 Z3 Hard-Constraints

Table 6.2 defines all the hard-constraints defined in 6.1.1 and implemented by z3.

Hard-Constraints		Meaning
$\bigvee_{(i,j)\|c_{i,j}\in C} y_{i,j}^{l,m} \iff x_{l,m}^{n}$	$\forall(l,m) \mid n_{l,m} \in N$	Each instance of the NSF is used if and only if there is at least one Capability that use it
$\sum_{(l,m)\|n_{l,m}\in N} bool\_to\_int(y_{i,j}^{l,m}) = 1$	$\forall(i,j) \mid c_{i,j} \in C$	Each instance of the same Capability must be used and allocated in a single instance of the available NSFs
$\sum_{j\|c_{i,j}\in c_i} bool\_to\_int(y_{i,j}^{l,m}) \leq 1$	$\forall(l,m) \mid n_{l,m} \in N$, $\forall i \mid c_i \in C$	All the instances of the same capability must be allocated in different instances of the same function. This means that in each instance of a function is possible to allocate only zero or one instance of the same capability
$\bigvee_{s\|h_s\in H} z_{l,m}^{s} \iff x_{l,m}^{n}$	$\forall(l,m)\|n_{l,m} \in N$	Each function instance is allocated in a host only if used
$\sum_{s\|h_s\in H} bool\_to\_int(z_{l,m}^{s}) \leq 1$	$\forall(l,m)\|n_{l,m} \in N$	If used, each instance of the function must be allocated in only an host
$\bigvee_{(l,m)\|n_{l,m}\in N} z_{l,m}^{s} \iff x_{s}^{h}$	$\forall s\|h_s \in H$	Each host is used if and only if there is at least one function which is allocated in it

$\sum_{\substack{(l,m)\mid n_{l,m}\in N \\ W_k^s}} w_{k,l} * bool\_to\_int(z_{l,m}^s) \leq$	$\forall s \mid h_s \in H, \forall k \in N_{0\to 3}$	The sum of all the resources required by the function to be allocated must be less or equal than the maximum supported by the physical server
$z_{l,m}^s \iff n_l \in N_s^h$	$\forall (l,m) \mid n_{l,m} \in N$	If set, the instance of an NSF can be allocated in the host if and only if the host supports it

Table 6.2: The Hard-Constraints of Z3

6.2.3 Z3 Soft-Constraints

Table 6.3 defines all the soft-constraints defined in 6.1.2 and implemented by z3. These constraints seek to optimize choices.

Soft-Constraints	Meaning
Soft($!x_j^n, w_j^n$, "cost")	The soft constraint to minimize the number of NSFs used. Each n_j has a cost w_j^n
Soft($!x_j^n, w_{k,j}$, "resource")	The soft constraint to minimize the use of n_j based on $w_{k,j}$ of the r_k chosen. If more resources are chosen, the respective weights are added up

Table 6.3: The Soft-Constraints of Z3

6.3 Gurobi formulation

In this section will be shown the formulas and notations used by Gurobi in the SAP module. These formulations allow to do the selection and optimization phase.

6.3.1 Gurobi Symbols

Table 6.4 defines all symbols constituting the elements and relationships used by Gurobi.

Symbols	Notations
C	Set of capabilities
$c_{i,j}$	The j-th instance of the i-th capability $\in C$
$x^c_{i,j}$	Binary variable, 1 if the instance $c_{i,j}$ is requested
N	Set of NSFs
$n_{l,m}$	The m-th instance of the l-th NSF $\in N$
$x^n_{l,m}$	Binary variable, 1 if the instance $n_{l,m}$ is used
$y^{l,m}_{i,j}$	Binary variable, 1 if the $c_{i,j}$ is supported and assigned to $n_{l,m}$
r_k	String variable, represents the resource; $r_k = \begin{cases} cost & \text{if } k = 0 \\ cpu & \text{if } k = 1 \\ ram & \text{if } k = 2 \\ disk & \text{if } k = 3 \\ bandwidth & \text{if } k = 4 \end{cases}$
$w_{k,l}$	Integer variable, it is the weight associated with the resource r_k of the function n_l
H	Set of hosts
w^n_l	Integer variable, it is the weight associated with the function n_l
h_s	The s-th host $\in H$
x^h_s	Binary variable, 1 if the h_s host is used
N^h_s	Set of NSFs supported by the h_s
$z^s_{l,m}$	Binary variable, 1 if the instance $n_{l,m}$ is allocated in h_s
L	Set of links
$l_{a,b}$	Binary variable, 1 if the host h_a and h_b are linked
W^s_k	Integer variable, it is the weight associated with the resource r_k of the host h_s
$W^l_{a,b}$	Integer variable, it is the weight associated with the link $l_{a,b}$
p_o	Integer variable, it is the priority assigned to the o-th Objective

Table 6.4: The symbols used by Gurobi

6.3.2 Gurobi Constraints

Table 6.5 defines all the constraints defined in 6.1.1 and implemented by Gurobi.

Number	Hard-Constraints		Meaning
1.a	$\sum_{(i,j)\|c_{i,j}\in C}^{K_1} y_{i,j}^{l,m} \leq K_1 * x_{l,m}^{n}$	$\forall(l,m) \mid n_{l,m} \in N$	Each instance of the NSF is used if and only if there is at least one Capability that use it
1.b	$x_{l,m}^{n} \leq \sum_{(i,j)\|c_{i,j}\in C} y_{i,j}^{l,m}$	$\forall(l,m) \mid n_{l,m} \in N$	The constraint 1.b is required because if the sum of the left side is null then the right side must also be null in the 1.a. Without this constraint the right side could have a value greater than 0
2	$\sum_{(l,m)\|n_{l,m}\in N} y_{i,j}^{l,m} = 1$	$\forall(i,j) \mid c_{i,j} \in C$	Each instance of the same Capability must be used and allocated in a single instance of the available NSFs
3	$\sum_{j\|c_{i,j}\in c_i} y_{i,j}^{l,m} \leq 1$	$\forall(l,m) \mid n_{l,m} \in N$ $\forall i \mid c_i \in C$	All the instances of the same capability must be allocated in different instances of the same function. This means that in each instance of a function is possible to allocate only zero or one instance of the same capability
4.a	$\sum_{s\|h_s\in H}^{K_2} z_{l,m}^{s} \leq K_2 * x_{l,m}^{n}$	$\forall(l,m)\|n_{l,m} \in N$	Each function instance is allocated in a host only if used

4.b	$x^n_{l,m} \leq \sum_{s\|h_s \in H}^{K_2} z^s_{l,m}$	$\forall (l,m) \| n_{l,m} \in N$	The constraint 4.b is required because if the sum of the left side is null then the right side must also be null in the 4.a. Without this constraint the right side could have a value greater than 0
5	$\sum_{s\|h_s \in H} z^s_{l,m} \leq 1$	$\forall (l,m) \| n_{l,m} \in N$	If used, each instance of the function must be allocated in only an host
6.a	$\sum_{(l,m)\|n_{l,m} \in N}^{K_3} z^s_{l,m} \leq K_3 * x^h_s$	$\forall s \| h_s \in H$	Each host is used if and only if there is at least one function which is allocated in it
6.b	$x^h_s \leq \sum_{(l,m)\|n_{l,m} \in N} z^s_{l,m}$	$\forall s \| h_s \in H$	The constraint 6.b is required because if the sum of the left side is null then the right side must also be null in the 6.a. Without this constraint the right side could have a value greater than 0
7	$\sum_{(l,m)\|n_{l,m} \in N} w_{k,l} * z^s_{l,m} \leq W^s_k$	$\forall s \mid h_s \in H, \forall k \in N_{0 \to 3}$	The sum of all the resources required by the functions to be allocated must be less or equal than the maximum supported by the physical server
8	$z^s_{l,m} \iff n_l \in N^h_s$	$\forall (l,m) \| n_{l,m} \in N$	If set, the instance of an NSF can be allocated in the host if and only if the host supports it

Table 6.5: The Hard-Constraints of Gurobi

6.3.3 Gurobi Multi-Objectives

Table 6.6 defines all the objectives defined in 6.1.2 and implemented by Gurobi.

Objectives	Meaning
$\text{Obj}\Big(\min\Big(\sum_{(l,m)\mid n_{l,m}\in N} n_{l,m} * w_{0,l}\Big), p_0\Big)$	It is the objective to minimize the $cost$ of the $n_{l,m}$s used. The objective has the priority p_0 chosen by the user
$\text{Obj}\Big(\min\Big(\sum_{(l,m)\mid n_{l,m}\in N} n_{l,m} * w_{1,l}\Big), p_1\Big)$	It is the objective to minimize the cpu of the $n_{l,m}$s used. The objective has the priority p_1 chosen by the user
$\text{Obj}\Big(\min\Big(\sum_{(l,m)\mid n_{l,m}\in N} n_{l,m} * w_{2,l}\Big), p_2\Big)$	It is the objective to minimize the ram of the $n_{l,m}$s used. The objective has the priority p_2 chosen by the user
$\text{Obj}\Big(\min\Big(\sum_{(l,m)\mid n_{l,m}\in N} n_{l,m} * w_{3,l}\Big), p_3\Big)$	It is the objective to minimize the $disk$ of the $n_{l,m}$s used. The objective has the priority p_3 chosen by the user
$\text{Obj}\Big(\min\Big(\sum_{(l,m)\mid n_{l,m}\in N} n_{l,m} * w_{4,l}\Big), p_4\Big)$	It is the objective to minimize the $bandwidth$ of the $n_{l,m}$s used. The objective has the priority p_4 chosen by the user
$\text{Obj}\Big(\min\Big(\sum_{(l,m)\mid n_{l,m}\in N} x^n_{l,m}\Big), p_5\Big)$	It is the objective to minimize the number of $n_{l,m}$s used. The objective has the priority p_5
$\text{Obj}\Big(\min\Big(\sum_{s\mid h_s\in H} x^h_s\Big), p_6\Big)$	It is the objective to minimize the number of h_ss used. The objective has the priority p_6

Table 6.6: The objectives of Gurobi

6.4 Comparison and final discussion

As seen in the previous sections, the symbols defined for Z3 (6.2.1) and Gurobi (6.3.1) are similar but have substantial differences. Z3 is a SMT solver and as such works mainly on boolean variables while Gurobi is an ILP solver and works with integer variables. This is a very important discrepancy because it entails a substantial difference in the formulation of the model and in the internal functioning of the tools. In fact, the constraints, although having the same meaning, have been defined differently for Z3 and Gurobi.

As first thing you can observe that the number is slightly different, this because z3 makes use also of logical formulas while Gurobi only of mathematical formulas.

To this purpose in z3 all the logical symbols appear as "$\bigvee$" and " $\iff$ " while in Gurobi the "$\sum$" appears and that is why in Gurobi there are additional formulas to express the concept of "if and only if".

As an instance, if we have the scenario composed by the following elements:

- 1 instance of the function N1;
- 2 instances of the capability C1. These two capabilities are supported by N1.

To express the 1st hard-constraint:

- in Z3:

  We have to impose the following variables:

  - $x^n_{1,1}$, boolean variable which represents the instance of the function N1;
  - $y^{1,1}_{1,1}$ and $y^{1,1}_{1,2}$, boolean variables which represent the two instances of the capability C1, assigned to N1.

  The formula 6.1 allows to impose the 1st constraint.

$$y^{1,1}_{1,1} \vee y^{1,1}_{1,2} \iff x^n_{1,1} \tag{6.1}$$

  In fact, as you can see in Fig. 6.2, if at least one variable between "$y^{1,1}_{1,1}$" and "$y^{1,1}_{1,2}$" is TRUE then necessarily also "$x^n_{1,1}$" is TRUE and equally, if "$x^n_{1,1}$" is TRUE then at least one between "$y^{1,1}_{1,1}$" and "$y^{1,1}_{1,2}$" must be TRUE. In all other cases the variables shall be FALSE.

$y_{1,1}^{1,1}$	$y_{1,2}^{1,1}$	$y_{1,1}^{1,1} \vee y_{1,2}^{1,1}$
T	T	T
T	F	T
F	T	T
F	F	F

Figure 6.1: Truth table of OR

$y_{1,1}^{1,1} \vee y_{1,2}^{1,1}$	$x_{1,1}^{n}$	$y_{1,1}^{1,1} \vee y_{1,2}^{1,1} \iff x_{1,1}^{n}$
T	T	T
T	F	F
F	T	F
F	F	T

Figure 6.2: Truth table of equivalence

- in Gurobi:

  We have to impose the following variables:

  - $x_{1,1}^{n}$, binary variable which represents the instance of the function N1;
  - $y_{1,1}^{1,1}$ and $y_{1,2}^{1,1}$, binary variables which represent the two instances of the capability C1, assigned to N1.

  The formula 6.2 allows to impose part of the 1st constraint.

$$y_{1,1}^{1,1} + y_{1,2}^{1,1} <= 2 * x_{1,1}^{n} \tag{6.2}$$

  In fact, if at least one between "$y_{1,1}^{1,1}$" and "$y_{1,2}^{1,1}$" is 1 then also "$x_{1,1}^{n}$" is set to 1 in order to respect the inequality. The number "2" which represents the cardinality of the instances it is important because if both the instances of capability are equal to 1 then the right part of the mismatch must be equal to 2 and the variable "$x_{1,1}^{n}$" is obliged to be 1. If the cardinality multiplying the variable is not present, then the inequality cannot be satisfied.

But what if both variables "$y_{1,1}^{1,1}$" and "$y_{1,2}^{1,1}$" are equal to 0? "$x_{1,1}^{n}$" can assume any value. In fact, whether it is 0 or 1, the inequality is respected. We do not want this behavior, in fact if there are no instances of capabilities assigned to a function instance then the function instance must not be used.

For this reason the formula 6.3 has been added:

$$x_{1,1}^{n} <= y_{1,1}^{1,1} + y_{1,2}^{1,1} \tag{6.3}$$

In this way, if the two variables are null then so does "$x_{1,1}^{n}$".

The differences continue and are centralized with the phase of optimization:

- Z3 does not support a true optimization mechanism. What it can do is to define *"soft-constraints"*, which are some constraints that it tries to respect. Each constraint has a weight that penalizes the solution.

  Each constraint is associated with a group. In the event that there are more ties associated with the same group then Z3 tries to satisfy all the constraints, blending and considering them as a single constraint.

  If different groups are specified there is no a precise way to assign a priority to one or the other and this is certainly a limitation of the tool because there is no way to better manage the multi objectives.

- Gurobi supports a real mechanism to manage objectives. It is able to manage multiple-objectives, it allows to associate a weight and a priority to each objective and to meet them hierarchically, blending them or mixing the two approaches.

  The hierarchical mode is the one implemented in the framework. Each objective has a priority. This allows Gurobi to solve as much as possible starting from the one with the highest priority and gradually going down, trying to reach the others without degrading the solution. This is definitely a strong point of the tool.

Another substantial difference concerns the goodness and speed of execution. Z3 is a SAT solver, Gurobi is an ILP solver. This difference is substantial. Gurobi produces better results in a shorter time and this is because the model to be solved is completely different.

The important differences analyzed so far have led to the choice of Gurobi as solver of the framework. Its speed of execution, the goodness of the solution and the ability to manage optimally multiple objectives contributed definitively to the choice.

6.5 Further development

As described in the sections above, the framework, and therefore the *SAP* module, at this time is able to determine how many and which instances of network functions are necessary, to satisfy the instances of capability received as input, and to allocate them among the physical hosts available.

The module is able to optimize the choice according to the parameters defined by the user, in order to:

- reduce the number of functions;
- reduce the cost of the functions;
- reduce the resources used by the functions.

  E.g. cpu, ram, disk and bandwidth;
- reduce the number of hosts;

All these objectives are managed simultaneously hierarchically according to the priority that the user associates them. In fact we are talking about *multi-objectives*.

In the future, these features could be improved and new ones could be added. For example, we could think of improving the host model. In fact, right now the hosts are simply listed.

We could think of linking them together via links and to assign a weight to each of them, weight representing its latency. This could lead to a further goal in the optimization phase: Choose hosts in such a way that they are linked with lower weight links so as to reduce latency.

Chapter 7

Implementation

7.1 Installation Guidelines

This chapter will show and describe the procedure for using the framework in the correct way. All additional tools to download and install will be shown in order to run the framework correctly.

7.1.1 JAVA JDK 8 SE

The *Java Platform SE (Standard Edition)* lets to develop and deploy Java applications on desktops and servers. To achieve this result it offers the rich user interface, performance, versatility, portability, and security that today's applications require. The *JDK (Java Development Kit)* includes tools useful for developing and testing programs written in the Java programming language and running (it includes also the JRE, Java Runtime Environment) on the Java platform.

At the following links you can find further documentation and download the development environment:

- https://www.oracle.com/technetwork/java/javase/documentation/index.html, to which it is possible to find all the documentation;
- https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, to which it is possible to download the JDK. For compatibility with other tools, **Java JDK SE 8** was chosen;

After downloading the JDK, it is necessary to set the environment variable **JAVA_HOME** to the folder where the JDK was unpacked.

E.g., ***{JAVA_HOME}*** = *"C:/ProgramFiles/Java/jdk1.8.0/221"*

7.1.2 Apache Ant

Apache Ant is a Java library and command-line tool for automating software build processes. Ant uses XML to describe the code build process and its dependencies. By default,

the XML file is named build.xml.

At the following links you can find further documentation and download the development environment:

- https://ant.apache.org/manual/index.html, to which it is possible to find all the documentation about the line **1.10.x**. ;
- https://ant.apache.org/bindownload.cgi, to which it is possible to download the binary distribution. For compatibility with other tools, the release **1.10.6** was chosen. Make sure you choose the development line 1.10.x and not 1.9.x as the Java8 library was chosen;

After downloading the Apache Ant, it is necessary to set the environment variable **ANT_HOME** to the folder where it was installed.

E.g., ***{ANT_HOME}*** = *"C:/ProgramFiles/Java/apache-ant-1.10.6"*

7.1.3 Apache Tomcat

Apache Tomcat is an open source web server from Apache. It implements Java Servlet, Java Server Pages (JSP), Java EL and WebSocket and provides an environment in which JAVA code can run as well as HTTP web server. Its servlet container is Catalina.

At the following links you can find further documentation and download it:

- http://tomcat.apache.org/tomcat-8.5-doc/index.html, to which it is possible to find all the documentation about the line **8.5.x**;
- https://tomcat.apache.org/download-80.cgi, to which it is possible to download the binary distribution. For compatibility with other tools, the release **8.5.35** was chosen.

After downloading the Apache Tomcat, it is necessary to set the environment variable **CATALINA_HOME** to the folder where it was installed.

E.g., ***{CATALINA_HOME}*** = *"C:/ProgramFiles/Java/apache-tomcat-8.5.35"*

It is also required to modify the **$CATALINA_HOME$/conf/tomcat-users.xml** file. Under the **tomcat-users** tag place the following code:

```
1 <role rolename="manager-gui"/>
2 <role rolename="manager-script"/>
3 <role rolename="admin"/>
4
5 <user username="tomcat" password="tomcat" roles="admin,manager-gui,
    ↪ manager-script"/>
```

Finally, you also need to edit the file **tomcat-build.xml** by changing the contents of the *"to_be_defined"* tag with the credentials defined before.

7.1.4 Neo4j

Neo4j is a NoSQL graph database management system. It uses graphs to represent data and the relationships between them. A graph is defined as any graphical representation that consists of vertices (shown by circles) and edges (shown with intersection lines). At each vertice or edge it is possible to associate any number of attributes. This tool is, therefore, particularly useful in debugging because it gives a clear and simplified view of the data.

At the following links you can find further documentation and download it:

- https://neo4j.com/docs/operations-manual/current/introduction/, to which it is possible to find all the documentation;
- https://neo4j.com/download/, to which it is possible to download the binary distribution. For compatibility with other tools, the release **3.5.x** was chosen.

After downloading Neo4j, it is necessary to set the environment variable **NEO4J_HOME** to the folder where it was installed.

E.g., ***{NEO4J_HOME}*** = *"C:/neo4j-community-3.5.5"*

To be able to start the Neo4j Server using the ant file in the framework, you must first enable the corresponding service: go to the **bin** folder and exec **neo4j** with the argument **install-service**.

7.1.5 Gurobi

Gurobi, whose name stands for its founders Zonghao **Gu**, Edward **Ro**thberg and Robert **Bi**xby, is a commercial state-of-the-art solver for mathematical programming. It is able to solve linear programming (LP), quadratic programming (QP), quadratically constrained programming (QCP), mixed integer linear programming (MILP), mixed-integer quadratic programming (MIQP), and mixed-integer quadratically constrained programming (MIQCP).

It was fouded in 2008 and supports a variety of programming and modeling languages.

At the following links you can find further documentation and download it:

- https://www.gurobi.com/documentation/quickstart.html, to which it is possible to find all the documentation;
- https://www.gurobi.com/downloads/, to which it is possible to download the binary distribution.

After downloading Gurobi, in order to use it, a licence must be obtained and it is necessary to proceed with a series of configurations according to your operating system.

Windows

- Set the environment variable **GUROBI_HOME** to the folder where it was installed.

  E.g., ***{GUROBI_HOME}*** = *"C:/gurobi811/win64"*

- Add the bin path to the **PATH** variable:

  E.g., ***PATH*** = *"C:/gurobi811/win64/bin"*

Linux

- Set the environment variable **GUROBI_HOME** to the folder where it was installed.

  1. *sudo nano /etc/environment*
  2. ***{GUROBI_HOME}*** = *"C:/gurobi811/linux64"*

- Add the bin path to the **LD_LIBRARY_PATH** variable:

  1. *sudo nano /.bashrc*
  2. ***{GUROBI_HOME}*** = *"C:/gurobi811/linux64"*
  3. *export* ***LD_LIBRARY_PATH*** = *$LD_LIBRARY_PATH:${GUROBI_HOME}/lib*
  4. *export* ***PATH*** = *$PATH:${GUROBI_HOME}/bin*

- You need to copy the file "libgurobi81.so" and *"libGurobiJni81.so"* to the folder **/usr/lib/**

7.2 Folders organization

In this chapter will be presented the packages containing the classes and methods that make up the framework. The framework includes also a detailed javadoc which describe all the classes and methods in detail.

Project folders are organized this way:

1. The **src** folder contains all the packages regarding the framework;

   In particular:

   (a) The package **it.polito.verifuse.main** contains the entry point of the framework;

   (b) The package **it.polito.verifuse.modules** contains the classes of the two modules;

   (c) The package **it.polito.verifuse.rest** contains the classes constituting the REST API;

   (d) The packages **it.polito.verifuse.rest.framework.resources**, **it.polito.verifuse.rest.framework.service** and **it.polito.verifuse.rest.framework.db** contain the classes of the Server.

   (e) The package **it.polito.verifuse.rest.framework.client** contains the classes of the Client

   (f) The packages **it.polito.verifuse.factory** and **it.polito.verifuse.utility** contains the classes of utility used in the framework.

2. The **xsd** folder contains XML schemas representing the data structures used;

3. The **gen-src** folder will contain all data structures generated by the JAXB framework starting from the XML schemas;

4. The **lib** folder contains the libraries used by the framework;

5. The **test** folder contains the packages which the tests;

   In particular:

   (a) The package **it.polito.verifuse.framework.test** contains the tests regarding the two modules;

   (b) The package **it.polito.verifuse.rest.test** contains the tests regarding the web server.

6. The **WebContent** folder contains all the files of the web module;

7. The **build.xml** file allows to start the framework;

8. The **tomcat-build.xml** file allows to manage tomcat;

9. The **service-build.xml** file allows to manage the deployment of the Web REST Server;

10. The **neo4j-build.xml** file allows to manage neo4j;

11. The **config.xml** file which is a file of configuration.

7.3 REST API

The **REST** (REpresentational State Transfer) is an architecture for distributed systems that allows clients to exchange information with the server using HTTP protocol methods such as GET, POST, UPDATE, DELETE, etc. The information that is exchanged concerns the representation of the *resources* made available by the web server, so what is exchanged is the *representational state*[13].

REST APIs have been developed to provide the functionality of the framework outdoors. In particular, to allow the access to CAID and SAP modules in order to exploit their potential.

7.3.1 Service Design

The RESTful Web Server has been implemented following the paradigm: *Resources - Service - Database*.

- **Resources**: represent all the resources of the service to which identifies the allowed HTTP methods; It is implemented by the VerifuseResources class.
- **Service**: contains the implementation of the HTTP methods and then what action to perform. As the framework deals with the management of CAID and SAP modules, to manage them better, the management part of the services has been divided into two. Therefore, the service is implemented by the CAIDService and SAPService classes.
- **Database**: contains the database and so all the data structures used by the service. Each class representing the service has its own dedicated database, so they are represented by the CAIDDB and SAPDB classes.

As it is possible to see in Fig. 7.1.

[13]The REST principles were discussed by Roy Thomas Fielding in his doctoral dissertation in 2000 [20].

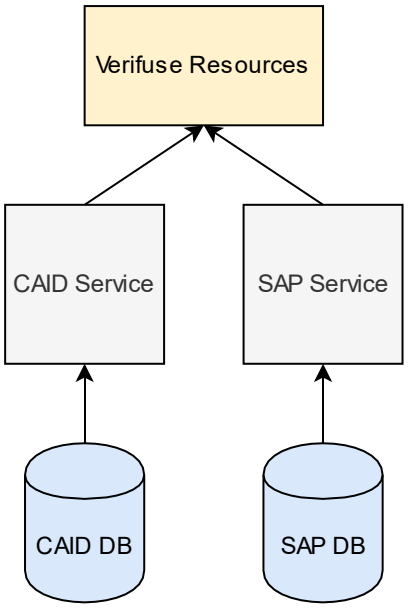

Figure 7.1: Implementation design

Resources

The verifuse framework provides the following resources so that it is possible to interact with the main resource (the framework itself) and its modules **CAID** and **SAP**.

Resources	Meaning	Relative URLs
verifuse	The framework (main resource)	verifuse
CAID	The CAID module of the framework	verifuse/caid
capabilities	The capabilities in the CAID	verifuse/caid/capabilities
{capabilityID}	The capability uniquely identified by "capabilityID"	verifuse/caid/capabilities/{capabilityID}
capabilitiesRequired	The capabilities required by the policies in the CAID	verifuse/caid/capabilitiesRequired
{capabilityID}	The policy uniquely identified by "capabilityID"	verifuse/caid/capabilitiesRequired/{capabilityID}
policies	The policies in the CAID	verifuse/caid/policies
{policyID}	The policy uniquely identified by "policyID"	verifuse/caid/policies/{policyID}
templates	The templates in the CAID	verifuse/caid/templates
{templateID}	The template uniquely identified by "templateID"	verifuse/caid/templates/{templateID}
SAP	The SAP module of the framework	verifuse/sap
capabilitiesInstances	The instances of the capabilities in the SAP	verifuse/sap/capabilitiesInstances
{capabilityID, capabilityInstance}	The instance of capability uniquely identified by the couple "capabilityID, capabiltyInstance"	verifuse/sap/capabilitiesInstances/{capabilityID_capabilityInstance}
hosts	The hosts in the SAP	verifuse/sap/hosts
{hostID}	The host uniquely identified by "hostID"	verifuse/sap/hosts/{hostID}
nsfs	The nsfs in the SAP	verifuse/sap/nsfs
{nsfID}	The nsf uniquely identified by "nsfID"	verifuse/sap/nsfs/{nsfID}
result	The result of the selection and placement in the SAP	verifuse/sap/result

Figure 7.2: Resources

Verifuse Resources

The **Verifuse Resources** is the class which manage the REST resources. It responds to requests for resources, previously defined in the design, by HTTP methods and manages them by generating appropriate responses.

Below you can see some code fragments that show how they work.

```
@Path("/verifuse")
@Api(value = "/verifuse")
public class VerifuseResources {
   public UriInfo uriInfo;
   private CAIDService caidService;
   private SAPService sapService;

   public VerifuseResources() {
   }

   /**
   * The constructor, receives the uriInfo by the context: JAX-RS
      ↪ runtime.
   *
   * @param uriInfo the uriInfo
   */
   public VerifuseResources(@Context UriInfo uriInfo) {
      this.uriInfo = uriInfo;
      try {
         this.caidService = new CAIDService(uriInfo.getBaseUriBuilder()
            ↪ .clone().path("verifuse").path("caid"));
         this.sapService = new SAPService(uriInfo.getBaseUriBuilder()
            ↪ .clone().path("verifuse").path("sap"));
      } catch (Exception e) {
         throw new InternalServerErrorException();
      }
   }
}
```

This piece of code shows how the allocation, by constructor, of the main resource takes place and how the services are initialized.

```
/**
* @return the main resource: Verifuse
*/
@GET
@ApiOperation(value = "getVerifuse", notes = "Reads the main resource."
)
@ApiResponses(value = {
   @ApiResponse(code = 200, message = "OK"),
})
@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
public it.polito.verifuse.rest.jaxb.verifuse.VerifuseResponse
    ↪ getVerifuse() {
   try {
      // Create the URI
      UriBuilder root = uriInfo.getAbsolutePathBuilder();
      UriBuilder caid = root.clone().path("caid");
      UriBuilder sap = root.clone().path("sap");

      // Create a representation of the main resource
      it.polito.verifuse.rest.jaxb.verifuse.VerifuseResponse verifuse =
          ↪ new it.polito.verifuse.rest.jaxb.verifuse.VerifuseResponse();

      // Add the URI
      verifuse.setSelf(root.toTemplate());
      verifuse.setCaid(caid.toTemplate());
      verifuse.setSap(sap.toTemplate());
      return verifuse;
   } catch (Exception e) {
      throw new InternalServerErrorException();
   }
}
```

Instead, this piece of code shows how the **GET** method executed on the main resource is executed, producing the answer **VerifuseResponse**.

It is important to take into account that each method, combined with resources, is enclosed in a try-catch block so that, when some internal exception occurs, to the client is sent an *InternalServerErrorException* message and not the whole stack trace (as default) that inexorably shows the internal structure of the server, which must be kept hidden from the client.

As it is possible to see in the previous code fragment, the instruction:

```
@ApiOperation(value = "getVerifuse", notes = "Reads the main
                        resource.")
@ApiResponses(value = { @ApiResponse(code = 200, message = "OK"),
                        })
```

It allows the documentation of REST operations through **Swagger**.

It is possible to post the resources using Swagger taking care to correctly specify the namespace within the xml file.

The SwaggerUI is available at the link:

http://localhost:8080/verifuse/webapi/swagger.json.

Services

As the framework deals with the management of **CAID** and **SAP** modules, to manage them better, the management part of the services has been divided into two.

- **CAIDService** handles all service requests addressed to the module *CAID*
- **SAPService** handles all service requests addressed to the module *SAP*

As an instance, if we want to access the resource representing a single **Policy** of the CAID module the request is handled by the **VerifuseResources** in the following fragment:

```
/**
* @param policyID of the policy
* @return the Policy
*/
@GET
@Path("/caid/policies/{policyID}")
@ApiOperation(value = "getCAIDPolicy", notes = "Reads a single policy."
)
@ApiResponses(value = {
   @ApiResponse(code = 200, message = "OK"),
   @ApiResponse(code = 404, message = "Not Found"),
})
@Produces({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
public it.polito.verifuse.rest.jaxb.caid.PolicyResponse
    ↪ getCAIDPolicy(@PathParam("policyID") String policyID) {
   try {
      it.polito.verifuse.rest.jaxb.caid.PolicyResponse policy =
          ↪ caidService.getPolicy(policyID);
      if (policy==null) {
      throw new NotFoundException();
      }
      return policy;
   } catch (NotFoundException e) {
      throw new NotFoundException();
   } catch (Exception e) {
      throw new InternalServerErrorException();
   }
}
```

The request is then executed inside the **CAIDService** in the following way:

```
/**
* Gets a Policy which 'policyID' is passed as argument.
*
* @param policyID the policyID
* @return the Policy
*/
public it.polito.verifuse.rest.jaxb.caid.PolicyResponse getPolicy(String
    ↪ policyID) {
   return db.getPolicy(policyID);
}
```

The **CAIDService** accesses the database to satisfy the request.

Database

Each service in order to perform its functions, gives access to its own database containing all data structures.

- **CAIDDB** it is database dedicated to the service *CAIDService*
- **SAPDB** it is database dedicated to the service *SAPService*

Both were developed implementing the Singleton pattern. Each instance is then created only on first access (which coincides with the creation of the service) and appears to be thread safe as the internal method is synchronized and a double check is made that avoids 2 threads (which arrive first simultaneously) to allocate it both. As it is possible to see in the following code fragment:

```
// static volatile variable single_instance
private static volatile SAPDB SAPDB_instance = null;

/**
* Static method to create instance of Singleton class in a synchronized
    ↪ way
* @return the SAPDB singleton
* @throws GeneralException
*/
public static SAPDB getInstance() throws GeneralException
{
   if (SAPDB_instance == null) {
      synchronized (SAPDB.class) { //Check for the second time.
      //if there is no instance available... create new one
         if (SAPDB_instance == null) {
            SAPDB_instance = new SAPDB();
         }
      }
   }
   return SAPDB_instance;
}
```

The constructor is private and this contributes to the Singleton pattern, making possible the creation of the instance only from the inside. The internal variable **SAPDB_instance** is declared as *volatile*, in this way is not possible for another thread in Java to see half initialized state of the variable. All the write will happen on volatile *SAPDB_instance* before any read and this helps to get it initialized by a single thread.
This, synchronization and double internal control, before the call to the constructor, ensure that the class is thread safe and that a single instance is allocated.

```
/**
* Private constructor restricted to this class itself, it is used only a
    ↪ time.
* It is thread safe because it is synchronized when used.
*/
private SAPDB() {
   mapCapabilitiesInstances = new ConcurrentHashMap<String,
      ↪ it.polito.verifuse.rest.jaxb.sap.CapabilityInstanceResponse>();
   mapHosts           = new ConcurrentHashMap<String,
      ↪ it.polito.verifuse.rest.jaxb.sap.HostResponse>();
   mapNsfs          = new ConcurrentHashMap<String,
      ↪ it.polito.verifuse.rest.jaxb.sap.NSFResponse>();
}
```

The constructor is responsible for initializing the data structures used by the database. They are implemented as *ConcurrentHashMap* to ensure access to concurrent threads.

```
// synchronized data structures
private ConcurrentHashMap<String,
    ↪ it.polito.verifuse.rest.jaxb.sap.CapabilityInstanceResponse>
    ↪ mapCapabilitiesInstances = null;
private ConcurrentHashMap<String,
    ↪ it.polito.verifuse.rest.jaxb.sap.HostResponse> mapHosts = null;
private ConcurrentHashMap<String,
    ↪ it.polito.verifuse.rest.jaxb.sap.NSFResponse> mapNsfs = null;
```

7.3.2 API Description

The Verifuse web service is a web service composed by two modules:

- CAID (CApability IDentifier) which has the task of analyzing policies and selecting the capabilities necessary to satisfy them and to send the capabilities to the ADP module.
- SAP (Selection And Placement) which has the task to receive the instances of the capability from the ADP module and to select the necessary network functions taking care also to allocate them in the physical server.

Verifuse API

Resource	Verb	Req. Body	Query params	Meaning	Verb		Resp. Body
verifuse	GET			Get the main resource	200	OK	Verifuse

Figure 7.3: Operation on Verifuse

This is the main API for the Verifuse Web Service. It provides links to reach the other modules.

Example request

- **GET** http://localhost:8080/verifuse/webapi/verifuse
- Accept: **APPLICATION_XML**;

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: XML file which allows navigation between resources.

```
<?xml version="1.0" encoding="UTF-8"?>
<VerifuseResponse xmlns="http://www.verifuse.polito.it/rest">
   <self>http://localhost:8080/verifuse/webapi/verifuse</self>
   <caid>http://localhost:8080/verifuse/webapi/verifuse/caid</caid>
   <sap>http://localhost:8080/verifuse/webapi/verifuse/sap</sap>
</VerifuseResponse>
```

CAID API

The following APIs represent the module **CAID**, by accessing this resource it is possible to use its functionality.

Resource	Verb	Req. Body	Query params	Meaning	Verb		Resp. Body
verifuse/caid	GET			Get a description of the module	200	OK	CAID
verifuse/caid/capabilities	GET		authorID: String	Get all the capabilities defined in the framework that are wrote by the {authorID} or all if it is not specified	200	OK	capabilities
verifuse/caid/capabilities/{capabilityID}	GET			Get the capability {capabilityID}	200	OK	capability
					404	Not Found	
verifuse/caid/capabilitiesRequired	GET			Get the capabilities required by the policies. The capabilities are selected by the module	200	OK	capabilities
verifuse/caid/capabilitiesRequired/{capabilityID}	GET			Get the capability {capabilityID}	200	OK	capability
					404	Not Found	
verifuse/caid/policies	GET			Get the policies	200	OK	capabilities
	POST	policy		Post the policy	201	Created	capability
					400	Bad Request	
					403	Forbidden	reason
verifuse/caid/policies/{policyID}	GET			Get the policy {policyID}	200	OK	policy
					404	Not Found	
	PUT	policy		Update the policy	201	Created	policy
					204	No Content	
					400	Bad Request	
					403	Forbidden	reason
					404	Not Found	
	DELETE			Delete the policy {policyID}	204	No Content	
					404	Not Found	

Figure 7.4: Operation on CAID (1)

Resource	Verb	Req. Body	Query params	Meaning	Verb		Resp. Body
verifuse/caid/templates	GET			Get the templates	200	OK	templates
	POST	template		Post the template	201	Created	template
					400	Bad Request	
					403	Forbidden	reason
verifuse/caid/templates/{templateID}	GET			Get the template {templateID}	200	OK	
					404	Not Found	template
	PUT	template		Update the template	201	Created	template
					204	No Content	
					400	Bad Request	
					403	Forbidden	reason
					404	Not Found	
	DELETE			Delete the template {templateID}	204	No Content	
					404	Not Found	

Figure 7.5: Operation on CAID (2)

As an example, in the following way it is possible to access the representation of the module:

Example request

- **GET** http://localhost:8080/verifuse/webapi/verifuse/caid
- Accept: **APPLICATION_XML**;

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: XML file which allows navigation between resources.

The body content of the response is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<CAIDResponse xmlns="http://www.verifuse.polito.it/rest/caid">
   <self>http://localhost:8080/verifuse/webapi/verifuse/caid</self>
   <capabilities>http://localhost:8080/verifuse/webapi/verifuse
      ↪ /caid/capabilities</capabilities>
   <capabilitiesRequired>http://localhost:8080/verifuse/webapi/verifuse
      ↪ /caid/capabilitiesRequired</capabilitiesRequired>
   <policies>http://localhost:8080/verifuse/webapi/verifuse
      ↪ /caid/policies</policies>
   <templates>http://localhost:8080/verifuse/webapi/verifuse
      ↪ /caid/templates</templates>
</CAIDResponse>
```

This allows to access the other resources contained in the module:

- **capabilities** which contains all the static capabilities that are defined in the module.
  - allows only the **GET** operation in order to be able to access the list.
- **capabilitiesRequired** which allows access to the key function of the module. Identify the *capabilities* that are needed to meet current policies in the db.
  - allows only the **GET** operation; Analyze existing policies and return identified capabilities.
- **policies** representing the policies present in the module.
  - **GET** to take the list which contains all the policies expressed;
  - **POST** to put a new **Policy** in the db.

Example request

* **POST** http://localhost:8080/verifuse/webapi/verifuse/caid/policies
* Content-Type: **APPLICATION_XML**;
* Content: the **Policy**.

Example response

* 200: **OK**
* Content-Type: **APPLICATION_XML**;
* Content: the **PolicyResponse**, a new rapresentation of the resource containing the Self URI.

Each *Policy* is characterized by a **policyID** that identifies it, this allows to perform the following operations on the resource at the address:
URI: http://localhost:8080/verifuse/webapi/verifuse/caid/policies/{policyID}

* **GET** to access it;
* **UPDATE** to modify it or to create it if does not exist;
* **DELETE** to delete it.

- **templates** representing the templates present in the module to which policies may refer.
  - **GET** to take the list which contains all the templates available;
  - **POST** to put a new **Template** in the db.

Example request

* **POST** http://localhost:8080/verifuse/webapi/verifuse/caid/templates
* Content-Type: **APPLICATION_XML**;
* Content: the **Template**.

Example response

* 200: **OK**
* Content-Type: **APPLICATION_XML**;
* Content: the **TemplateResponse**, a new rapresentation of the resource containing the Self URI.

Each *Template* is characterized by an **templateID** that identifies it, this allows you to perform the following operations on the resource at the address:
URI: http://localhost:8080/verifuse/webapi/verifuse/caid/templates/{templateID}

* **GET** to access it;
* **UPDATE** to modify it or to create it if does not exist;
* **DELETE** to delete it.

SAP API

The following APIs represent the module **SAP**, by accessing this resource it is possible to use its functionality.

Resource	Verb	Req. Body	Query params	Meaning	Verb		Resp. Body
verifuse/sap	GET			Get a description of the module	200	OK	SAP
verifuse/sap/ capabilitiesInstances	GET		capabilityID: string	Get the instances of the capability{capabilityID} or of all the instances if {capabilityID} is null	200	OK	Found capabilities instances
	POST	capability Instance		Post the instance of the capability	201	Created	capabilityInstance
					400	Bad Request	
					403	Forbidden	reason
verifuse/sap/ capabilitiesInstances/ {capabilityID_ capabilityInstance}	GET			Get the instance of the capability {capabilityID_ capabilityInstance}	200	OK	capabilityInstance
					404	Not Found	
	DELETE			Delete the instance of the capability {capabilityID_ capabiltiyInstance}	204	No Content	
					404	Not Found	
verifuse/sap/hosts	GET			Get the hosts	200	OK	hosts
	POST	host		Post the host	201	Created	host
					400	Bad Request	
					403	Forbidden	reason
verifuse/sap/hosts/ {hostID}	GET			Get the host {hostID}	200	OK	host
					404	Not Found	
	PUT	host		Update the host	201	Created	host
					204	No Content	
					400	Bad Request	
					403	Forbidden	reason
					404	Not Found	
	DELETE			Delete the host {hostID}	204	No Content	
					404	Not Found	

Figure 7.6: Operation on SAP (1)

Resource	Verb	Req. Body	Query params	Meaning	Verb		Resp. Body
verifuse/sap/nsfs	GET			Get the nsfs	200	OK	nsfs
	POST	nsf		Post the nsf	201	Created	nsf
					400	Bad Request	
					403	Forbidden	reason
verifuse/sap/nsfs/ {nsfID}	GET			Get the nsf {nsfID}	200	OK	nsf
					404	Not Found	
	PUT	nsf		Update the nsf	201	Created	nsf
					204	No Content	
					400	Bad Request	
					403	Forbidden	reason
					404	Not Found	
	DELETE			Delete the nsf {nsfID}	204	No Content	
					404	Not Found	
verifuse/sap/result	GET		VNFmustBe SupportedByHost: boolean priorityCpu: integer priorityRam: integer priorityDisk: integer priorityBandwidth: integer priorityCost: integer	Get the result based on the optimization chosen	200	OK	
					404	Not Found	result

Figure 7.7: Operation on SAP (2)

As an example, in the following way it is possible to access the representation of the module:

Example request

- **GET** http://localhost:8080/verifuse/webapi/verifuse/sap
- Accept: **APPLICATION_XML**;

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: XML file which allows navigation between resources.

The body content of the response is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<SAPResponse xmlns="http://www.verifuse.polito.it/rest/sap">
   <self>http://localhost:8080/verifuse/webapi/verifuse/sap</self>
   <capabilitiesInstances>http://localhost:8080/verifuse/webapi
      ↪ /verifuse/sap/capabilitiesInstances</capabilitiesInstances>
   <hosts>http://localhost:8080/verifuse/webapi/verifuse
      ↪ /sap/hosts</hosts>
   <nsfs>http://localhost:8080/verifuse/webapi/verifuse /sap/nsfs</nsfs>
   <result>http://localhost:8080/verifuse/webapi/verifuse
      ↪ /sap/result</result>
</SAPResponse>
```

This allows to access the other resources contained in the module:

- **capabilitiesInstances** representing the instances of the capabilities that must be assigned to the functions.
  - **GET** to take the list which contains all the instances of the capabilities; It is possible specify the query param $\{capabilityID\}$ so as to request all instances of capability with that identifier. If it is *NULL* then they are all returned.

    Example request

    * **GET** http://localhost:8080/verifuse/webapi/verifuse/sap/capabilitiesInstances
    * Accept: **APPLICATION_XML**;
    * Query-Param: {**capabilityID**}**:string**.

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: the **CapabilitiesInstancesResponse** which contains a list of the capability instances selected.

– **POST** to put a new **CapabilityInstance** in the db.

Example request

- **POST** http://localhost:8080/verifuse/webapi/verifuse/sap/capabilitiesInstances
- Content-Type: **APPLICATION_XML**;
- Content: the **CapabilityInstance**.

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: the **CapabilityInstanceResponse**, a new rapresentation of the resource containing the Self URI.

Each *CapabilityInstance* is characterized by an **ID** that identifies it. It consists of the capability (*capabilityID*) and the instance (*capabilityInstance*) identifier: {**capabilityID_capabilityInstance**}, which allows you to perform the following operations on the resource at the address:
URI: http://localhost:8080/verifuse/webapi/verifuse/sap/capabilitiesInstances /{capabilityID_instance}

- **GET** to access it;
- **DELETE** to delete it.

• **hosts** representing the hosts to which the NSFs can be allocated.

– **GET** to take the list which contains all the hosts available;

– **POST** to put a new **Host** in the db.

Example request

- **POST** http://localhost:8080/verifuse/webapi/verifuse/sap/hosts
- Content-Type: **APPLICATION_XML**;
- Content: the **Host**.

Example response

* 200: **OK**
* Content-Type: **APPLICATION_XML**;
* Content: the **HostResponse**, a new rapresentation of the resource containing the Self URI.

Each *Host* is characterized by an **hostID** that identifies it, this allows you to perform the following operations on the resource at the address:
URI: http://localhost:8080/verifuse/webapi/verifuse/sap/hosts/{hostID}

* **GET** to access it;
* **UPDATE** to modify it or to create it if does not exist;
* **DELETE** to delete it.

- **nsfs** representing the nsfs available in the system.
  - **GET** to take the list which contains all the nsfs available;
  - **POST** to put a new **Nsf** in the db.

Example request

* **POST** http://localhost:8080/verifuse/webapi/verifuse/sap/nsfs
* Content-Type: **APPLICATION_XML**;
* Content: the **Nsf**.

Example response

* 200: **OK**
* Content-Type: **APPLICATION_XML**;
* Content: the **NsfResponse**, a new rapresentation of the resource containing the Self URI.

Each *Nsf* is characterized by an **nsfID** that identifies it, this allows you to perform the following operations on the resource at the address:
URI: http://localhost:8080/verifuse/webapi/verifuse/sap/nsfs/{nsfID}

* **GET** to access it;
* **UPDATE** to modify it or to create it if does not exist;
* **DELETE** to delete it.

- **result** representing the result of the selection and placement of functions.
  - **GET** to ask the module SAP to calculate the result. It is possible to specify the following query params:
    - **VNFmustBeSupportedByHost**: boolean variable, true if a host must support the function so that it can be allocated;
    - **optimizeCpu**: boolean variable, true if we want select the functions in order to optimize the use of cpu;
    - **optimizeRam**: boolean variable, true if we want select the functions in order to optimize the use of ram;
    - **optimizeDisk**: boolean variable, true if we want select the functions in order to optimize the use of disk;
    - **optimizeBandwidth**: boolean variable, true if we want select the functions in order to reduce the use of bandwidth;
    - **optimizeCost**: boolean variable, true if we want select the functions in order to reduce the cost;

Example request

- **GET** http://localhost:8080/verifuse/webapi/verifuse/sap/result
- Accept: **APPLICATION_XML**;

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: the **Result**

7.3.3 Service Deployment

The Web Server has been implemented using the **Jersey** framework, which is the reference implementation of the *Java API for RESTful Web Services* (JAX-RS)[14].

The Jersey RESTful Web Services framework is open source and in addition to implementing the reference standards, makes available its own additional features and utilities to further simplify RESTful service and client development.

The web server has been packaged and deployed on **Tomcat** which acts as a container. The *Tomcat Server* is an open source web server from Apache. It implements Java Servlet, Java Server Pages (JSP), Java EL and WebSocket and provides an environment in which JAVA code can run as well as HTTP web server. Its servlet container **Catalina** has allowed the deployment of the **Verifuse REST Web Server** making it reachable by clients through the HTTP procotol.

The web server is portable (it has been tested in windows and linux environments) thanks to the cross-platform nature of the java language.

In addition to the server part (and therefore the backend part) a REST client has also been developed in order to test the various services.

The test implementation was possible thanks to the use of the **JUnit** library.

[14]The *JAX-RS* refers to the documentation *JSR 311* written by the *Sun Microsystems* for the purpose of drawing up specifications for the implementation of the REST paradigm in Java [21]. The new standard *JAX-RS 2.0* refers to the documentation *JSR 339*. The *Jersey RESTful Web Services framework* supports both standards.

Chapter 8

Evaluation

A series of tests have been performed in order to evaluate the scalability and the time of execution of the framework. Tests were performed for both modules: CAID and SAP.

The tests were carried out on a machine having the following characteristics:

- **Operating Systems**: Windows 10 Education N;
- **CPU**: Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz;
- **Architecture**: x64;
- **RAM**: 8,00 GB Dual-Channel DDR3 @ 798MHz.

8.1 Tests on CAID

A series of tests were performed on the CAID module in order to test its scalability. Therefore, the purpose of the tests is to progressively increase the number of incoming policies (they start from one and ending with one million) in order to verify the execution time.

Test results can be analyzed in the Fig. 8.1, they are very positive. Indeed, high time values (3.6 hours) occur for one million ($10 * 10^5$) of policies. While for values less than one hundred thousand the execution time is less than one second. This last thing can be better observed in Fig. 8.2, which represents a zoom on the first part of the graph Fig. 8.1.

Analyzing in detail both graphs, we observe the following situations:

- Between **0 and 1.000** policies, the execution time is almost constant and is about **40 ms**;
- Between **1.000 and 10.000** policies, the execution time slowly increases up until it reaches **1 s**;
- Between **10.000 and 500.000** policies, the execution time begins to grow until it reaches **thirty minutes**;

- After 500.000 policies, the execution time grows faster and faster. During the final test, **one million** policies were analyzed in **three and a half hours** (132.040.686 ms).

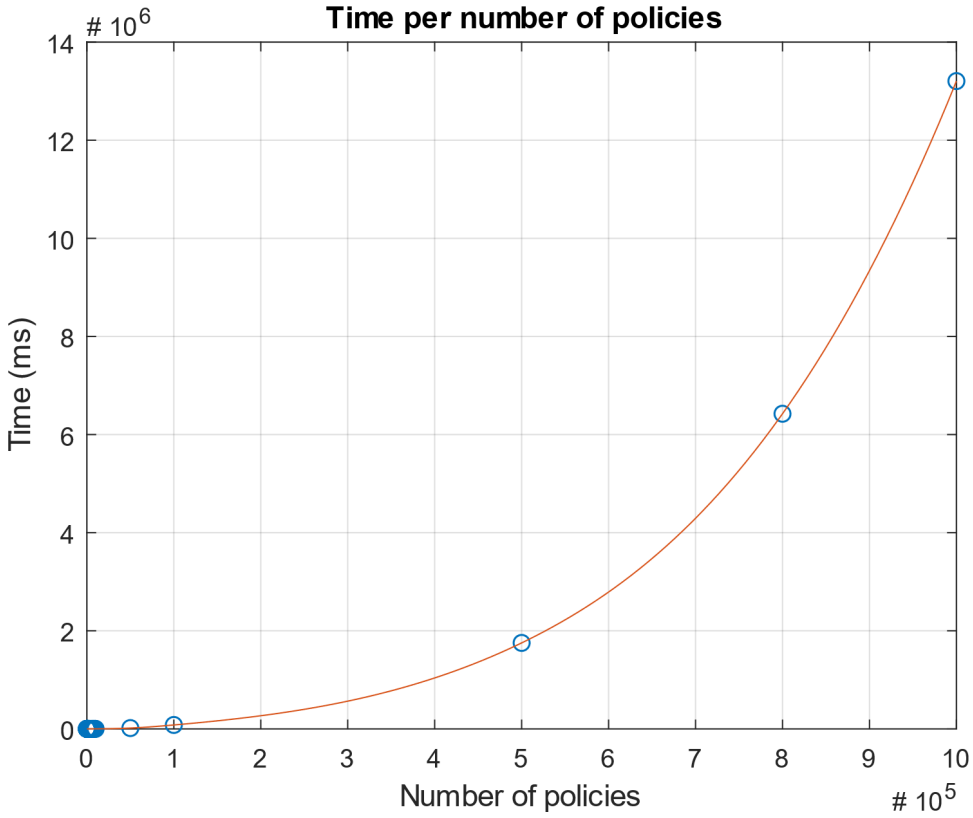

Figure 8.1: Time per number of policies

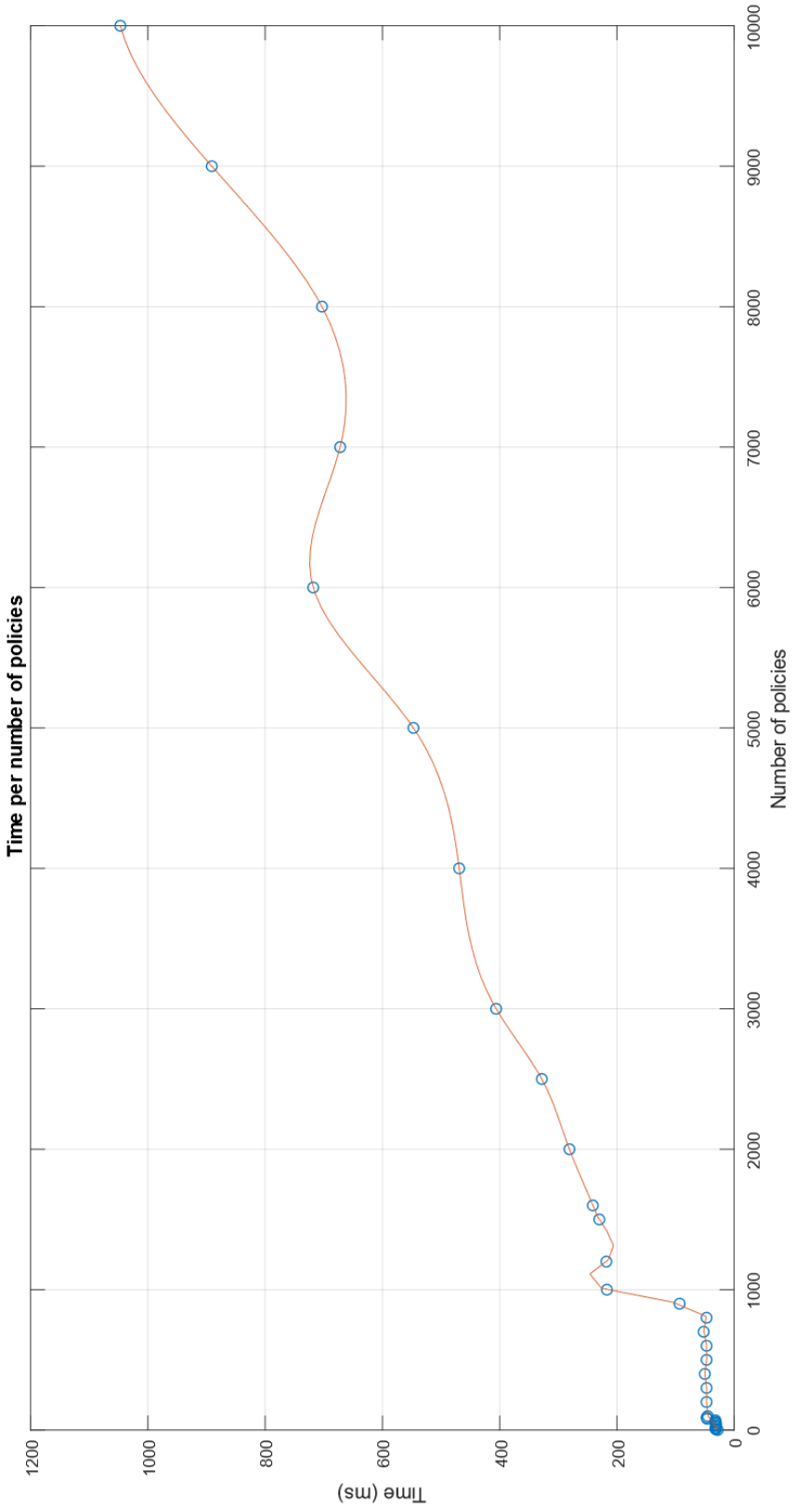

Figure 8.2: Time per number of policies, zoom

8.2 Tests on SAP

A series of tests were performed on the SAP module in order to test its scalability. Therefore, the purpose of the tests is to progressively increase the number of incoming instances of capabilities, network functions and hosts in order to monitor the execution time. The tests demonstrated that the execution time is more sensitive to the number of functions and less to the number of instances of capabilities and hosts, as shown below.

Capabilities = variable, NSFs = constant, Hosts = constant

The results in Fig. 8.3 have been obtained by varying the number of instances of capabilities and keeping the number of functions and hosts constant equal to one. The idea is to choose two capabilities and increase the number of their instances needed more and more.

Analyzing in detail the graph with the data, we can observe that the trend is linear, in particular we can distinguish the following situations:

- Between **0 and 2.000** capabilities, execution is very fast and requires about **50 ms**;
- After 2.000 capabilities, the execution is fast and linear. During the final test, the module receiving as input **one million** capabilities took only **47 s** (46.857 ms).

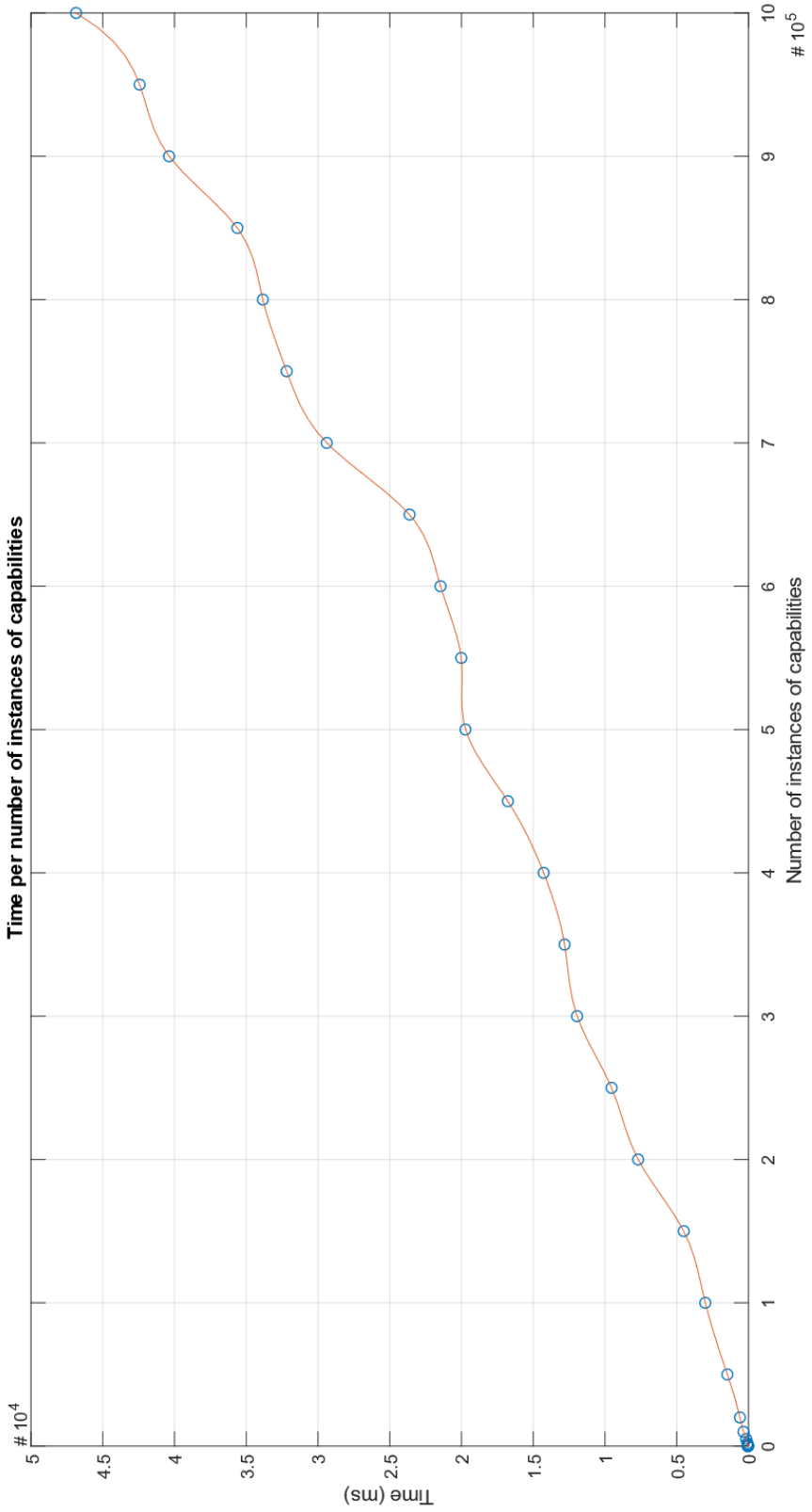

Figure 8.3: Time per number of capabilities

Capabilities = constant, NSFs = variable, Hosts = constant

The impact of the number of the NSFs in the framework is shown in the Fig. 8.4. This graph has been obtained by varying the number of function and keeping the number of instances of capability and hosts constant equal to one. It is important to note that the tests have been performed by increasing the copies of the same function, this allows to evaluate the performance of gurobi in the best way as the tool is forced to work in the worst situation.

Examining the graph of the Fig. 8.4 and Fig. 8.5, which represents an enlargement of the first time intervals, we can distinguish the following situations:

- Between **0 and 700** functions, the execution time is below **1 s**;
- Between **800 and 1.000** functions, the execution time is maximum of **2 s** (2.034 ms);
- Between **1.000 and 3.000** functions, the execution time is maximum of **10 s**;
- Between **4.000 and 10.000** functions, the execution time keeps constant around **25 s**;
- Between **10.000 and 40.000** functions, the execution time increases softly, coming to **12 m** (702.577 ms);
- After 40.000 functions, the execution time grows faster and faster. During the final test, **ninety thousand** policies were analyzed in **almost three hours** (10.659.112 ms).

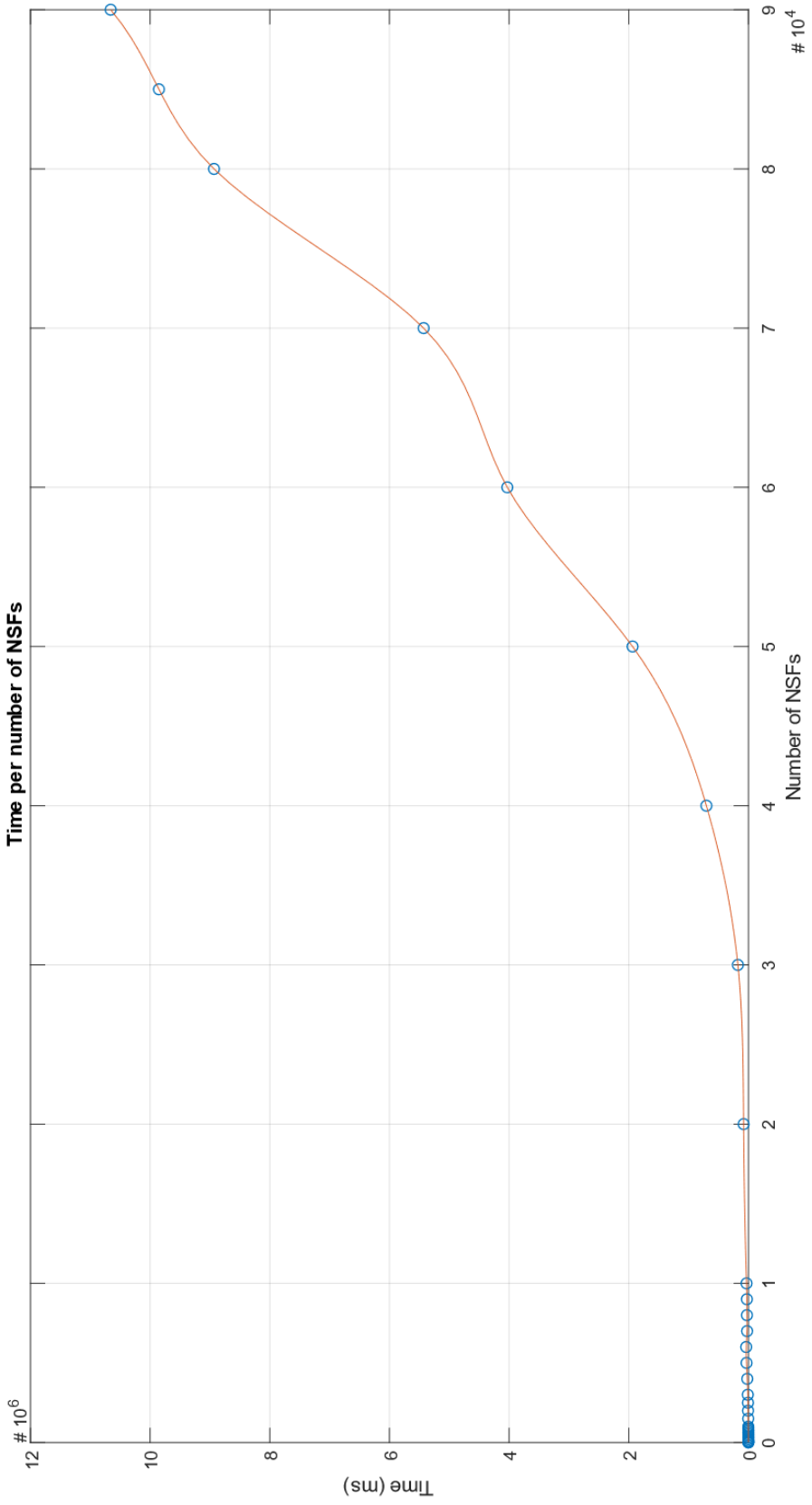

Figure 8.4: Time per number of nsfs

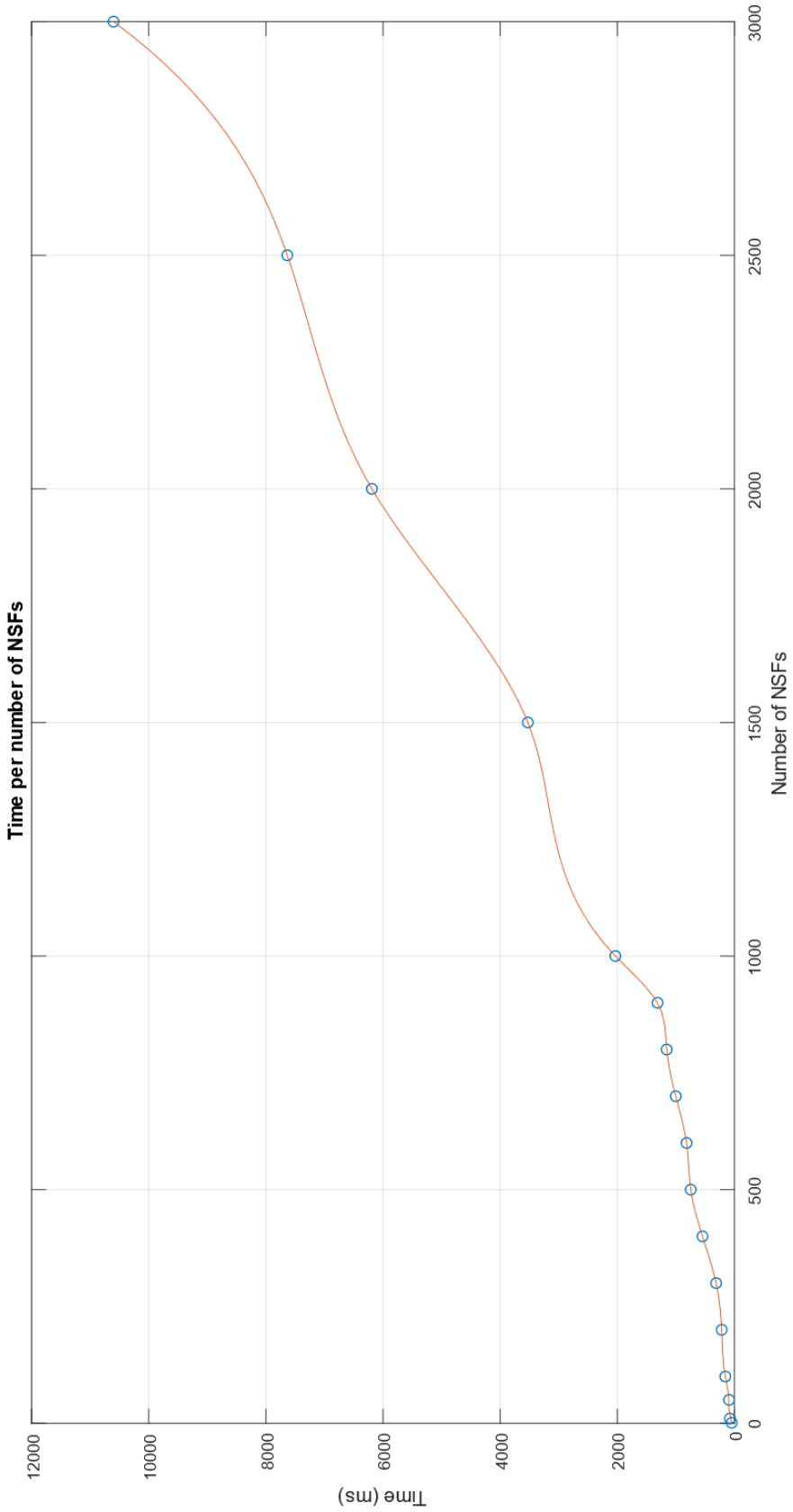

Figure 8.5: Time per number of nsfs, zoom

Capabilities = constant, NSFs = constant, Hosts = variable

The results in Fig. 8.6 have been obtained by varying the number of hosts and keeping the number of instances of capability and the number of functions constant equal to one.

Examining accurately the Fig. 8.6 and Fig. 8.7, we can note the following cases:

- Between **0 and 600** hosts, the execution time is below **100 ms**;
- Between **600 and 900** hosts, the execution time is below **200 ms**;
- Between **1.000 and 3.000** hosts, the execution time grows softly and is below **1 s** (812 ms);
- Between **3.000 and 6.000** hosts, the execution time continues to grow very slowly until about **2 s** (2261 ms);
- Between **6.000 and 10.000** hosts, the execution time begins to grow, coming to about **7 s** (6837 ms);
- After 10.000 hosts, the execution time grows faster and faster. During the last test, with **three hundred thousand** hosts the time is **almost three hours** (9.724.580 ms).

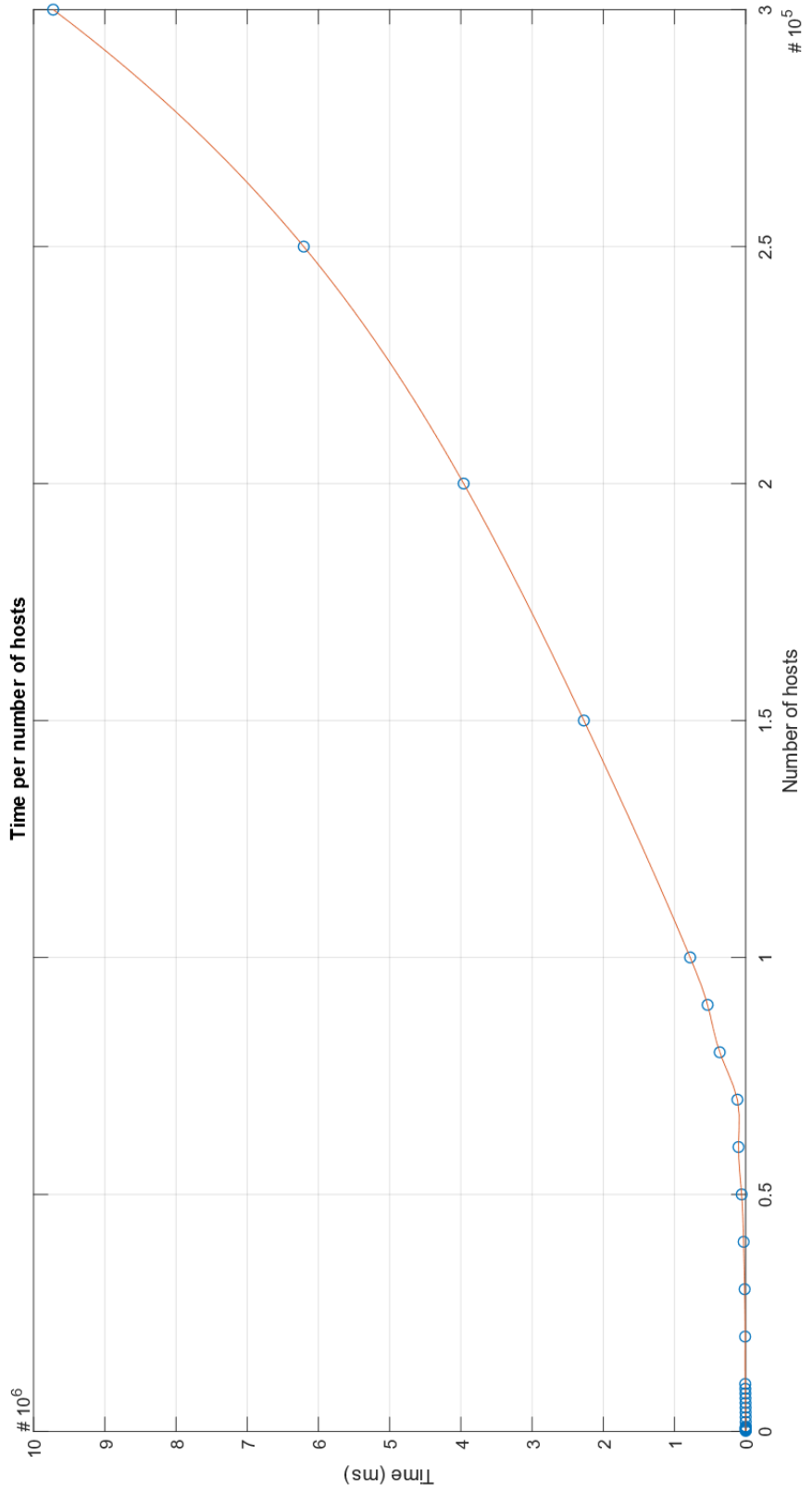

Figure 8.6: Time per number of hosts

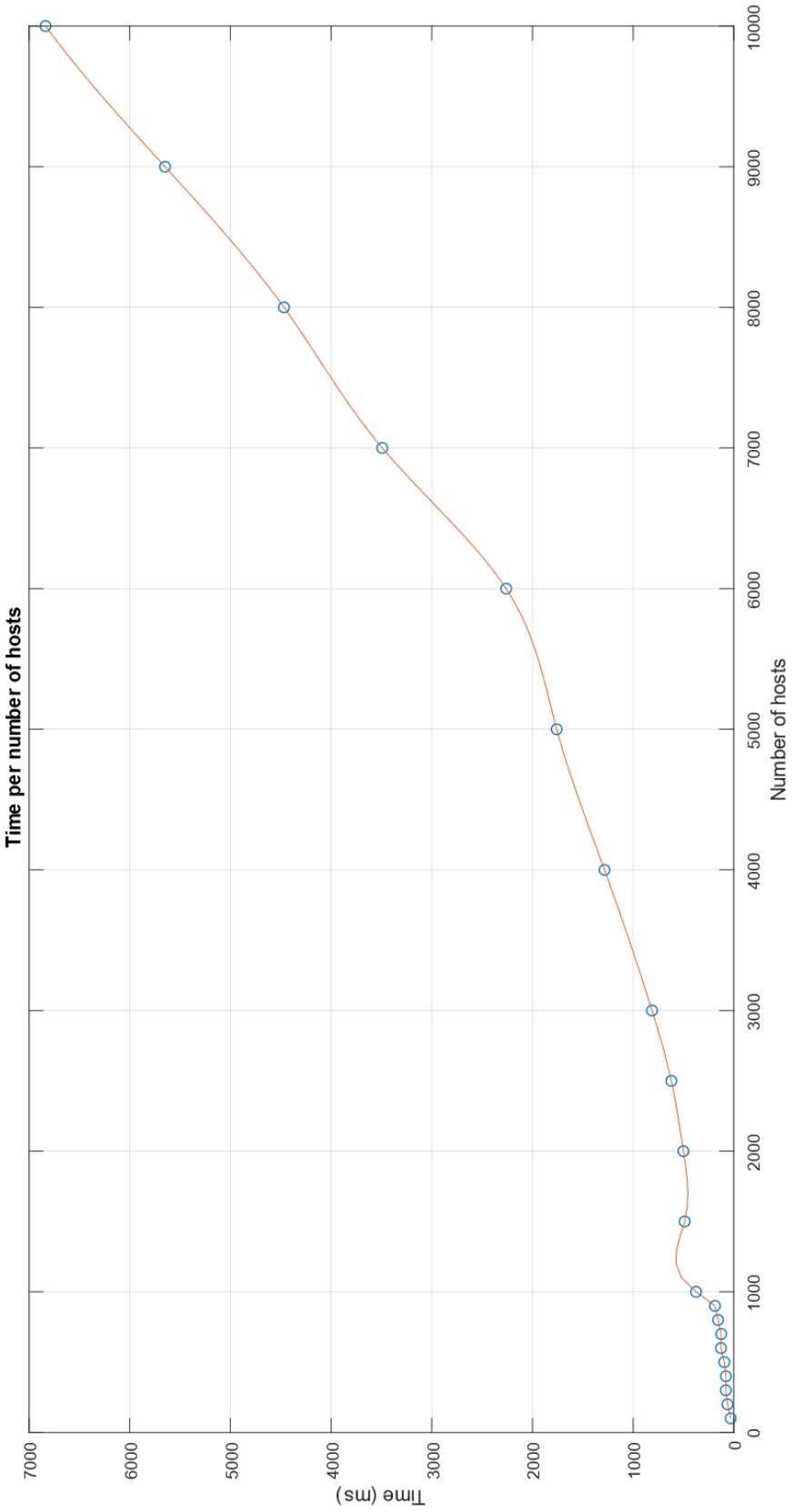

Figure 8.7: Time per number of hosts, zoom

Chapter 9

Conclusion

The development of this thesis required an in-depth theoretical study during the first phase. A study in which much attention has been paid to the works in literature regarding networks, security and policies. Study that led to the choice of the **Capability** as a fundamental concept around which to develop the framework. Unique concept, mentioned in literature but which has never been used in this way. The idea of using capabilities as a means of merging policies and network functions seems to be a winning idea and the excellent test results confirm it. The idea of capability allows to identify the functions independently from the vendor. In fact, two functions support the same capacity, for example packet filter, if they are able to fulfill this task by their internal features.

Then, starting from the concept of capability it was possible to model the **Catalogue of Functions** and the **Policy Repository**. The catalog is complete with all kinds of information regarding the available functions as hardware, software ed generic information while the policy repository contains policies expressed through HPL. This last has been modeled to be expandable in the future. In fact, although the framework currently works mainly on HPL, it has been prepared to contain also the policies expressed by MPL.

As last thing the physical **hosts** have been modeled, they are the server where to allocate the functions. In the future it is possible to improve the model, we could add some connections between the hosts in such a way as to consider the latency during the selection and optimization phase.

All of these models have been built using the **XML language**, which allows to create very precise and extended models giving also the possibility to express constraints. This language is well supported by Java thanks to the use of the **JAXB framework** that allows an easy marshal and unmarshal, from and to java classes.

The framework, through the modules **CAID** and **SAP**, analyzes the policies in income and identifies how much and which network functions are necessary and arranges to allocate them between the hosts available. The identification of capabilities, starting with policies and the choice of functions followed by their allocation, are the key points of the framework. These last steps are carried out through the use of an external tool: **Gurobi**.

Gurobi is a **mathematical solver** that solves arithmetic equations quickly and efficiently. The choice of values to assign to variables can be constrained and optimized based on objectives (as an instance, to maximize a value). Multi-objective management, performance and speed of execution are the key points that make it perfect for the framework. The tests carried out confirm this choice.

Tests of scalability have been performed to evaluate the performance of both the CAID module and the SAP module. The tests, despite having been performed on an old-generation laptop, produced excellent food for thought. The CAID module has proven that it can get the necessary capabilities in a very good time, analyzing one million policies took only three hours. The SAP module, on the other hand, seemed insensible to the number of incoming instances of capability (it produces a very good solution in forty-seven seconds, receiving one million of capabilities in input) while it was very dependent on the number of functions in the catalog.

Therefore, thesis work has achieved the goal of developing the **Verifuse** framework. The framework, through the CAID and SAP modules, provides everything you need to define and analyse policies, allowing you to create your own catalog of functions and your physical host infrastructure. After choosing the optimization criterion, Verifuse automatically selects which network functions are needed to meet the policies, or better the capabilities. The framework also provides a **REST API** interface to enable remote use. Verifuse, therefore, has been developed to support network administrators who every day with difficulty must first create policies, for various users, and then decide which network functions are needed. Verifuse automates all of this. Choose functions, starting with policy analysis, and allocate them among the available hosts. The choice is optimized, it is the user who defines the parameters and the priorities. The modules scale very well and have been designed in such a way as to be independent and to be easily **expandable** in the future.

Bibliography

[1] ETSI GS NFV. "ETSI GS NFV 003 V1.4.1, Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV", August 2018. [Online]. Available: https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV%20003v1.4.1%20-%20GS%20-%20Terminology.pdf.

[2] ETSI GS NFV. "GS NFV 002 V1.1.1, Network Functions Virtualisation (NFV); Architectural Framework.", October 2013. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf.

[3] Open Networking Foundation. "OpenFlow Switch SpecificationVersion 1.5.1 (Protocol version 0x06)", March 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[4] J. Highsmith et al. "Manifesto for Agile Software Development", 2001. [Online]. Available: https://agilemanifesto.org/.

[5] J. Halpern et C. Pignataro. "Service Function Chaining (SFC) Architecture", , rfc-7665, October 2015. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc7665.txt.pdf.

[6] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry and S. Waldbusser. "Terminology for Policy-Based Management", rfc-3198, November 2001. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc3198.txt.pdf.

[7] B. Moore, E. Ellesson, J. Strassner and A. Westerinen. "Policy Core Information Model – Version 1 Specification", rfc-3060, February 2001. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc3060.txt.pdf.

[8] D. D. Clark and D. R. Wilson. "A Comparison of Commercial and Military Computer Security Policies", ieee-6234890, April 27-29, 1987. [Online]. Available: https://ieeexplore.ieee.org/document/6234890.

[9] D. C. Robinson, M. S. Sloman. "Domains: a new approach to distributed system management", ieee-26694, September 14-16, 1988. [Online]. Available: https://ieeexplore.ieee.org/document/26694.

[10] R. Yavatkar, D. Pendarakis and R. Guerin. "A Framework for Policy-based Admission Control", rfc-2753, January 2000. [Online]. Available: https://tools.ietf.org/pdf/rfc2753.pdf.

[11] Y. Barthal, A. Mayer, K. Nissim and A. Wool. "Firmato: A Novel Firewall Management Toolkit", February 1999.

[12] C. Basile, A. Lioy, C. Pitscheider, F. Valenza and M. Vallini. "A novel approach for integrating security policy enforcement with dynamic network virtualization", April 2015.

[13] N. Bjørner, L. Moura, L. Nachmanson and C.Wintersteiger. *Microsoft Research*. ”Programming Z3”. [Online]. Available: https://theory.stanford.edu/~nikolaj/programmingz3.html.

[14] J. Moffett and M. Sloman. ”Policy hierarchies for distributed systems management,” *Selected Areas in Communications, IEEE Journal on, vol. 11, no. 9, pp. 1404-1414*, December 1993.

[15] M. Vallini, C. Basile, C. Pitscheider and F.Valenza. ”SECURED deliverable D4.1 - Policy specification”, March 2015.

[16] C. Basile, A. Lioy, C. Pitscheider, F.Valenza and M. Vallini. ”A novel approach for integrating security policy enforcement with dynamic network virtualization”, April 2015.

[17] G. Booch, J. Rumbaugh and Ivar Jacobson. ”The Unified Modeling Language User Guide, 2nd Edition”, May 2005.

[18] ISO 7498. ”The Basic Reference Model for Open Systems Interconnection”, iso-7498, October 1984.

[19] B. Volz, E. Cain, E. Kohler, J. Postel, M. Lottor, M. Accetta, R. Stewart, R. Adams, R. Nethaniel, R. Thomas and R. Ullmann. *IANA*. ”Service Name and Transport Protocol Port Number Registry”, May 2019. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

[20] Roy Thomas Fielding. ”Architectural Styles and the Design of Network-based Software Architectures”, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[21] M. Hadley, P. Sandoz. ”JAX-RS: Java™ API for RESTfulWeb Services”, September 2008. [Online]. Available: https://download.oracle.com/otn-pub/jcp/jaxrs-1.0-fr-eval-oth-JSpec/jaxrs-1.0-final-spec.pdf?AuthParam=1562875456_5646adf1e202745f73038a249a0bcc7f.

Index