Horizon Project

ANR call for proposals number ANR-08-VERS-010
FINEP settlement number 1655/08

Horizon - A New Horizon for Internet

WP3 - TASK 3.2: Piloting System

Institutions

GTA-COPPE/UFRJ
PUC-Rio
UNICAMP
Netcenter Informática ltda
LIP6 Université Pierre et Marie Curie
Telecom SudParis
Devoteam
Ginkgo Networks
VirtuOR

Abstract

This deliverable is composed of two parts. The first part presents the Piloting Plane concept, a networking plane that implements the notion of piloting of the management systems, as well as its initial design in the Horizon project. The piloting concepts are realised in the Horizon architecture with the help of Piloting Agents (PAs), which support the federation and distribution of self-piloting Autonomic Piloting Systems through its negotiation activities/tasks.

The second part shows the results obtained with a self-management system prototype, built to validate Piloting Plane presented on the first part. The prototype proposed is an intermediary between the control and management entities and the context of network and services. The piloting concepts of the system are realised with the help of multi-agents based on the Ginkgo Distributed Network Piloting System. Details about the testbed built to evaluate the proposed system are also presented.

Contents

List of Figures

Chapter 1

Introduction

Computer networks have become pervasive in society. Its services and technology have multiplied and they have become essential for the global economy. The management of them by humans is high costly and prone to failure. The simple automation of management through software components may worsen due to the wide variety of systems and protocols.

Autonomic networks [1, 2, 3] were proposed to deal with the problem of increasing complexity of telecommunications. They represent a specific topic in the area of autonomic computing [4], a term coined by IBM, intended to deal with complexity by enabling systems to self-manage themselves. This concept is bio-inspired by the autonomic nervous system that carries out the regulation of body in the face of changes in the environment without a conscious control. In the scenario of networks, simple tasks of configuration, optimization, disaster recovery and security are achieved by the network itself, leaving administrators free to more complex tasks such as setting policies and goals.

Autonomic networks are also within the context of cognitive networks [5], whose design is based on tools of artificial intelligence and cognitive systems. The term "cognitive networks" was inherited from cognitive radios, although the cognitive networks extrapolate to different kinds of networks. An essential part of the cognitive networks is the knowledge plane [6], a system which enables the network to assemble itself given high level instructions, reassemble itself as requirements change, automatically discover when something goes wrong, and automatically fix a detected problem.

Nowadays is also advocated the approach of pluralism of architectures for the future Internet over the one-size-fits-all TCP/IP [7]. The new approach defines that network providers should be splitted in service and infrastructure providers [8] and proposes the use of virtualization [9]. Users request network services from the service providers, which instantiate virtual networks over

the substrate provided by the infrastructure providers. Each virtual network can have its own protocols and configurations, in accordance with the objectives of the service running on it, and must have isolation, i.e., the operation of virtual networks does not cause interference between them, although they are on the same infrastructure. To allocate resources for such networks in an optimal, robust and secure way is a challenge due the complexity of the problem [10, 11, 12].

The Horizon project aims to define and validate a new network architecture based on the principles of pluralism and the knowledge plane. To achieve these objectives, it is necessary to have a piloting plane where the decisions are made. This report presents the results obtained with a self-management system proposed by us, which is within the piloting plane, being an intermediary between the control and management entities and the context of network and services.

Our purpose is to apply the Piloting Plane concept, a networking plane that implements the notion of piloting of the management systems, as well as its initial design in the Horizon project, in a scenario of virtual networks through a multi-agent sytem. The piloting concepts are realised in the Horizon architecture with the help of Piloting Agents (PAs), which support the federation and distribution of self-piloting Autonomic Piloting Systems through its negotiation activities/tasks. Figure 1.1 illustrates some ideas that we intend to explore during the work, such as specialized agents to a role, like power and performance control; the existence of a hierarchy of agents; and the management based on service level agreement (SLA). At the current stage, the main issue addressed is to have the agents in the network core performing actions in accordance with changes in the context.

This report is organized as follows. The Chapter 2 shows the autonomic piloting sytems concepts. In Chapter 3 we discuss about piloting plane functions and requirements. The preliminar piloting plane design is presented in Chapter 4, while the piloting agents architecture are showed in Chapter 5. In Chapter 6, we show how the testbed, which contains the substrate and virtual networks used to validate the self-management system, has been built. The tools used for the testbed and the development of some preliminary experiments are also presented in this chapter. Chapter 7 contains the description and the models of the system in terms of multi-agents. The results of the final experiments are shown in Chapter 8. Chapter 9 presents the conclusions and the next steps of this work.

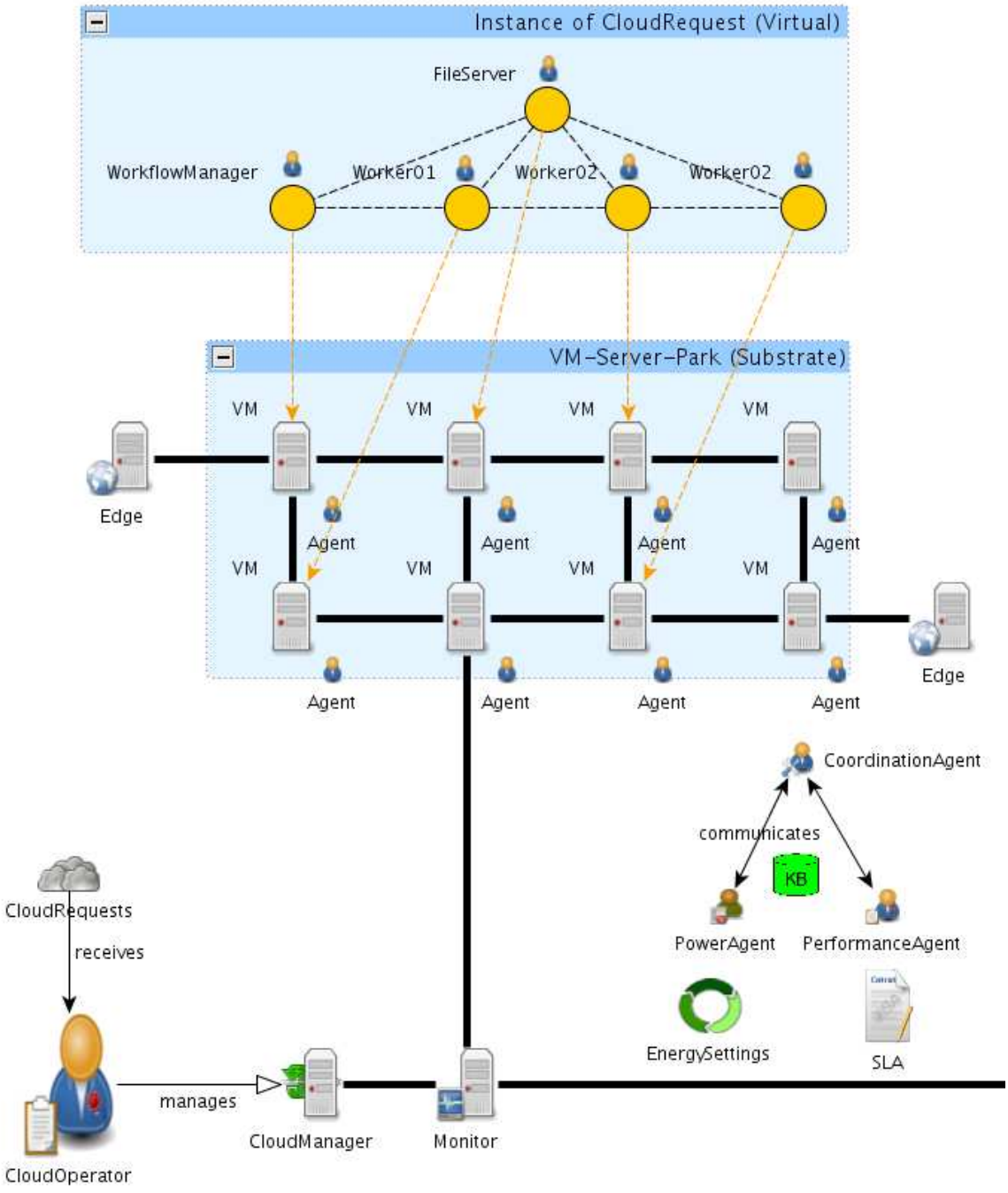

Figure 1.1: A multi-agent model to virtual network management.

Chapter 2

Autonomic Piloting Systems

Autonomic Piloting Systems, as initially defined in the IBM manifesto [13], have been defined as management systems of a single system. In networking, Autonomic Piloting Systems have to perform different management tasks covering various nodes, links and services. Due to the existence of several management standards, different protocols and different vendors, managing a network is much more complex than managing a single isolated system. Thus, it is not practical to devise a single autonomic control loop that autonomically adjusts all the FCAPS (fault, configuration, accounting, performance and security) aspects of a network. This means that we need to define one or more autonomic loops for each of those management aspects in order to simplify the design of each control loop. However, the operation of the network management system will depend on the interaction of all those control loops, which must ensure, amongst other key aspects that the network operates within normal parameters set by the business goals of the operators. Also, the decisions of a control loop may go against the objectives of another one. As an example, an autonomic security component may use a heavier encryption scheme to improve the security of the network, however this encryption scheme may require too much processing and bandwidth, reducing the maximum throughput of the network to a level below the performance dictated on the SLA. In networking, two sub-networks having different managers must interconnect. This requires that the protocols as well as the configuration of the network (i.e. security policies, QoS and SLAs) are compatible. If they are not, either a re-negotiation and re-configuration process is required, or translation services (gateways) must be installed in the border of the two networks. In order to solve those problems we are introducing a new system or plane, the Piloting Plane (PP), enabling cooperation of the various autonomic control loops ensuring their decisions are not orthogonal. This cooperation, or piloting, ensures that the overall optimization goals of each

autonomic component and control protocol are aligned with the goals and SLAs defined for the entire network, Piloting also means that autonomic management domains run by different operators or administrators are able to automatically adjust their configuration to accommodate the federation of networks. The need for a Piloting Plane (PP) arises from the deployment of several autonomic control loops with different administrators or management goals, which would not be able to interoperate without a set of translation, negotiation, federation and deployment functions. Thus, piloting deals with the meta-management of Autonomic Piloting Systems, that is, the deployment and reconfiguration of autonomic management control loops in order to allow their interoperation. This is achieved based on a set of high-level goals, defined for each of the managed network domains that form the piloted network. The PP ensures the interoperation of management systems, even though those systems use different set of high-level goals and management standards. This process may be accomplished through the negotiation of new SLAs and policies, the deactivation of conflicting management systems followed by the activation of other management systems, or the migration of such systems or parts of them within the piloted network. The entire piloting process is piloted by Piloting Policies, which dictate what are the compromises that each of the managed domains are willing to make for the sake of interoperability.

2.1 Architecture

The architecture of a virtual network environment can be composed of four planes:

- The Data plane forwarding the data.
- The Control plane where lies all the control algorithms necessary for monitoring the throughput, the security, the mobility, the reliability, etc.
- The Management plane in charge of all management features.
- The Piloting plane for feeding in real time the control planes.

However, one of the main changes, in relation with classical architectures, is the fusion of the control and management planes in just one plane to get a three planes autonomic architecture. A general view of these autonomic architectures is illustrated in Figure 2.1. The new paradigm in this architecture is the Piloting plane. This system could be seen as an aggregation

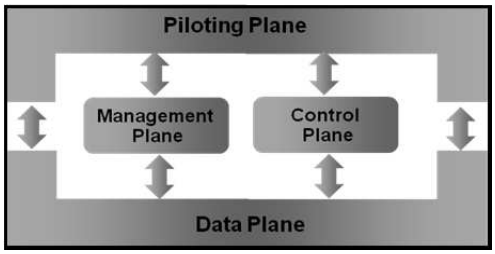

Figure 2.1: A general view of autonomic architectures.

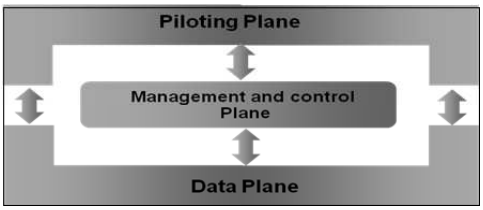

Figure 2.2: The piloting-oriented architecture.

of two specific sub planes: a knowledge plane and an orchestration plane. Unfortunately, the definitions of these two planes are not sufficiently precise. In our vision, the knowledge plane must be able to recover very often and very quickly the knowledge useful for feeding the control and management algorithms. In the same way, the orchestration plane is related to the conductor indicating in real time the tempo to his musicians. The reason of this integration (Figure 2.2) is the impossibility to dissociate these two planes during the implementation. The orchestration plane needs to have knowledge bases associated with the piloting intelligent process to be efficient. A first implementation of this piloting plane was described in reference [14] where the piloting plane is proposed as a meta-control plane. The papers [2] and [15] are describing more in details some parts of this architecture and finally reference [16] describes some examples.

First let us explain a little bit more in details the functions of these different planes. The data plane is in charge of transporting the data from one sender to one or several receivers. Since future home network is one of the scopes of this paper, we can assume that some home network elements may be virtualized (Internet box, set-up-box, etc.). For example, a physical Internet box can run several virtual boxes. Several virtual networks can be supported over a same physical network. In the same way, physical access points may support several virtual access points as well as controllers, boxes, and so on may support virtual instances. The management and control plane provides all management and control algorithms: failure detection,

diagnostic, security management, routing, flow control, security control, mobility, etc. Indeed, several algorithms for the same task could be available in the nodes. In this case, a choice is necessary depending on the context and the need of the network and the services. Information and knowledge need to have a syntactic presentation to permit the connection between different machines coming from various manufacturers. The Protégé platform supports two main ways of modelling ontologies via the Protégé-Frames and Protégé-OWL editors. Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema. Protégé is based on Java, is extensible, and provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development. The Piloting System has to drive the network through the control algorithms. For this purpose, the Piloting plane has to feed the management and control algorithms. As a summary, the Piloting plane has to orchestrate the Management and Control plane which configures the Data plane itself. Currently, in traditional home networks the values of the parameters are selected through information collected directly by the algorithms themselves. The advantage of the Piloting plane is the possibility to react in a short response time and in real time if necessary on the behavior of the management and control algorithms through the management and control plane. This piloting process aims to adapt the home network to new conditions and to take advantage of the piloting agent to alleviate the global system. A distributed intelligent agents system permits to achieve an adaptive control process due to the following two points: (1) each agent holds different processes (behaviour, dynamic planner and situated view) allowing to take the most relevant decisions at every moment; (2) the agents are implicitly cooperative in the sense that they use a situated view taking into account the state of the neighbours.

2.2 Definition of Piloting within Horizon

The purpose of the Piloting Plane is to govern and integrate the behaviours of the network in response to changing context and in accordance with applicable high level goals and policies. It supervises and integrates all other planes behaviour insuring integrity of the Future Internet management operations. The Piloting Plane can be seeing as a control framework into which any number of components can be plugged into or out in order to achieve the required functionality. The Piloting Plane would also supervise the optimisation and the distribution of knowledge within the Knowledge Plane to ensure that the required knowledge is available in the proper place at the proper time. This implies that the Piloting Plane may use either

very local knowledge to deserve a real time control as well as a more global knowledge to manage some long-term processes like planning. The Piloting Plane would host several Autonomic Piloting Systems (APSs). It is made up of one or more Piloting Agents (PAs), and a dynamic knowledge base consisting of a set of data models and ontologies and appropriate mapping logic. Each APS represents a set of virtual entities, which manage a set of virtual devices, sub-networks, or networks using a common set of policies and knowledge. The APSs access a knowledge base, which consists of a set of data models and ontologies. APSs can communicate and cooperate with each other, by the use of Behaviours, which act as stubs in the APS communication. A Piloting Plane can be federated with other Piloting Planes. The Piloting Plane acts as control workflow for all APSs ensuring bootstrapping, initialisation, dynamic reconfiguration, adaptation and contextualisation, optimisation, organisation, closing down of PAs. The Piloting Plane provides assistance for the Service Lifecycle Management, namely during the actual creation, deployment, activation, modification and in general, any operation related to the application services and/or management services. The APSs enables the following functions across the piloting plane:

Federation: the Federation enables a set of domains (APS Domain or Piloted Domain) to be combined into a larger domain (Piloted Domain or two-combined Piloted Domain) guided by common high level goals, while maintaining local autonomy. APS Federation: each APS is responsible for its own set of virtual and non-virtual resources and services that it governs as a domain. Federation enables a set of domains to be combined into a larger domain (Piloted Domain) guided by common high level goals, while maintaining local autonomy. Piloting Federation: two Piloted Domains federate to make a larger Piloted Domain. Here, the federation should take into account the different goals of two different Piloted Domains which would be combined and decide if federation will be possible.

Negotiation: in Horizon, negotiation can take place between autonomous entities with or without human intervention. APSs and PAs are the main entities that can be engaged in negotiations to achieve their goals. Each PA advertises a set of capabilities (i.e., services and/or resources) that it offers for use by other components in the Piloting Plane. The APSs performs negotiation between the APSs for the fulfilment of a specific SLA, defined by the operators of the managed piloted domains.

Distribution: the APSs provide communication and control services that enable tasks to be split into parts that run concurrently on multiple

PAs within the Piloting Plane.

Governance: each APS can operate in an individual, distributed, or collaborative mode (i.e. in federation). The APS collects appropriate monitoring data in order to determine if the virtual and non-virtual resources and services that it governs need to be reconfigured. High level goals, service requirements, context, capabilities and constraints are all considered as part of the decision making process.

System Views: the APSs are responsible for managing the system views that are stored and diffused using the knowledge plane. APSs will fetch the information required for their operation from the PAs as well as the services and resources through interfaces defined in the following.

2.3 Related Work

The autonomic concept was proposed to overcome the growth of complexity of current and future networks. There are some new concepts that deal with the autonomic aspect and the generic self-* properties. The idea is to facilitate the management and/or the control of networks regarding the growth of complexity. D. Clark in his paper Knowledge plane [6] recommends the construction of a new generation of networks able to "self-manage" themselves given high-level objectives without any human intervention. Clark's proposal of a knowledge plane in fact can be seen of a junction of the management, piloting and knowledge planes of the Horizon project. In the project we decided to separate those three planes in order to better tame the complexity. Other autonomic architectures like Focale proposed by Motorola [17] extend the knowledge plane concept by introducing high-level goals. Strassner's inference plane is another proposal for the piloting of networks. Other architectures were proposed through European FP7 program but again the objective is different from Horizon:

- The ANA Project aims at exploring novel ways of organizing and using networks beyond legacy Internet technology [18]. Their focus is mostly in protocols, not in management and piloting.
- The HAGGLE project deals with an innovative paradigm for autonomic opportunistic communication [19]. This project aims at developing a cross-layer network architecture exploiting intermittent connectivity by supporting opportunistic networking paradigm. In this project the needs for piloting are on the device level, while in Horizon we are dealing with the piloting of networks.

- The BIONETS project aims at a novel approach able to address the challenges of pervasive computing [20]. Learning from nature and society will allow to overcome heterogeneity and to achieve scalability via an autonomic peer-to-peer communication paradigm.

- The CASCADAS project has the objective of developing Component-ware for Autonomic, Situation-aware Communications and Dynamically Adaptable Services [21]. The project wants to propose an innovative architectural vision based on self-organized distributed components for autonomic and situation-aware communication.

- The Ambient Networks (AN) [22] is a FP6 project which envisaged the development of a software-driven network control infrastructure for wireless and mobile networks that will run on top of all current network physical infrastructures to provide a way for devices to connect to each other, and through each other to the outside world and to provide seamless service provisioning and roaming.

- 4D is a new architectural model for the Internet, where tasks are divided into 4 planes: Decision, Dissemination, Discovery and Data [23]. In 4D, the data plane is a simple plane, which only acts based on the configurations received by the decision plane. Decisions are taken based on the information recuperated by the Discovery plane, which constructs a view of the physical resources. Next, the decisions are sent to the Data plane using the Dissemination plane. The paper does not present any hard data, simulation or implementation to show the benefits of their architecture, however the authors argue that the main advantage of such an architecture is in the centralization of decisions into one single plane, removing the problems of multiple layers dealing with similar issues. 4D has two main differences with regards to the planes in Horizon. First, it fuses the management, control and piloting planes into one, however it does not describe a framework or design patterns to make the design of autonomic networks a tractable task. Second, it does not deal with the fact that we cannot rely on one single management entity, once each domain will be operated by a different organization. The piloting plane, however, accounts for this fact, allowing the negotiation and federation of different management domains.

2.4 Interaction of piloting, management and virtualization planes

An Horizon autonomic management architectural model consists of a number of distributed management systems described with the help of four abstractions - the four planes: Virtualization Plane (VP), Management Plane (MP), Knowledge Plane (KP), and Piloting Plane (PP). Together these distribute systems form a software-driven network control infrastructure that will run on top of all current virtual networks and service physical infrastructures to provide a way for devices or attachments to connect to each other, and through each other to the outside world and to provide seamless service provisioning. The PP will interact with the management plane through Behaviours, defined in Section 4.2. Each APSs will control one or more PAs. Each APSs will deal with the piloting issues related to the interoperation of the PAs overseen by the PA. For supporting these tasks, the APSs will require information from the virtualization plane, using some interfaces to fetch the required information. Further, APSs will aid in service deployment, starting up or closing down network and user services in the Horizon architecture. The APSs defines constraints on the deployment of new services, such as the set of virtual routers or networks where the service will be installed, as well as some of its execution parameters.

2.5 The responsibilities of the PP in the Horizon Architecture

The role of the Piloting Plane is to govern, dynamically adapt and optimize autonomic control loops in response to changing piloting–aware context and in accordance with applicable high-level goals and policies. It supervises and it integrates all other planes' behaviour, ensuring integrity of the management and control operations. Besides adapting the configuration of PAs, the Piloting Plane may also bootstrap and close down PAs when needed. The Piloting Plane can be thought of as a control framework into which any number of components can be plugged into in order to achieve the required functionality. The need for a Piloting Plane (PP) arises from the deployment of several autonomic control loops with different administrators or management goals, which would not be able to interoperate without a set of translation, negotiation, federation and deployment functions. Thus, piloting deals with the meta-management of Autonomic Piloting Systems, that is, the deployment and reconfiguration of autonomic management con-

trol loops in order to allow their interoperation. This is achieved based on a set of high-level goals, defined for each of the managed network domains that form the piloted network. The PP ensures the interoperation of management systems, even though those systems use different set of high-level goals and management standards. Ontology translation and mapping techniques between different data models based on the common information model can be used to create the common language upon participating entities that can negotiate, federate etc. This process of interoperation of management systems may be accomplished through the negotiation of new SLAs and policies, the deactivation of conflicting management systems followed by the activation of other management systems, or the migration of such systems or parts of them within the piloted network. The entire piloting process is governed by Piloting Policies, which dictate what are the compromises that each of the managed domains are willing to make for the sake of interoperability.

Chapter 3

Piloting Plane Functions and Requirements

The PP collaborates with all the other planes and as such it requires the following key functions and requirements:

- Take into account high-level goals (represented as policies, for example) and customer needs. It does not consider low-level technical details or high-level business details.
- The knowledge required for intra- and inter- domain piloting must be timely disseminated by the Piloting Plane and related components of the architecture.
- Depending on the service to be provisioned, the reaction time of the APSs must be constrained within certain boundaries in order to achieve the defined SLAs.
- The Piloting Plane must have interfaces to interact with the PAs (i.e. to cope with environment changes and conflicts).
- It should provide support for the PAs to define their dependencies on other components and services, as well as their expected operational conditions (e.g. based on policies describing their required services and virtual resources).
- The cooperation of the APSs from different domains requires the use of open protocols and standardized information and data models.
- The Piloting Plane must be aware of the state of virtual resources using the specific interface between virtualisation plane and piloting plane, and the information stored on the Knowledge Plane.

- Solve conflicts arising from orthogonal goals on different PAs. Thus, it must be capable to reach a compromise, allowing the overall system to achieve its purpose.
- It acts as control workflow for all PAs ensuring bootstrapping, initialisation, contextualisation, closing down of PAs. It also controls the sequence and conditions in which one PA invokes other PA in order to realize some useful functions (i.e., an piloting is the pattern of interactions between APSs).
- Enhancement and evolution: The Piloting Plane would allow relevant number of components to be plugged into or out in order to achieve the required functionality and without interruption of normal operation.

Chapter 4

Preliminar Piloting Plane Design

The piloting plane concept in the Horizon approach is composed by several Autonomic Piloting Systems (APSs). Each APS will be responsible for the interaction of several subordinated PAs. APSs will interact among themselves whenever necessary, in order to guarantee end-to-end SLAs and SLOs. The Piloting Plane governs the execution of the APSs. It acts as control workflow for all APSs ensuring bootstrapping, initialisation, dynamic reconfiguration, adaptation and contextualisation, optimisation, organisation, closing down of PAs. It also controls the sequence and conditions in which one PA invokes other PA in order to realize some useful functions (i.e., a piloting is the pattern of interactions between PAs). The Dynamic Planner is responsible for those tasks. Finally, each PA will interact with the Knowledge plane to fetch information regarding the controlled Behaviours (see Section 5). This information will vary for each PA, and will compose the situated view of the PA. The Situated View defines the information that is needed for the operation of an autonomic component and from where it must be collected. The Piloting Plane is made up of one or more Autonomic Piloting Systems (APSs): one per domain. Each controlled PA will have its associated Behaviour. The Behaviour is a wrapper for the PAs, providing the interfaces and functionality needed for the interaction with the PA. There is one APSd for each piloted domain, which will communicate with the APSs of other piloted domains to reach agreements, allowing the operation of the network as a whole. A federation of the Piloting Plane with other Piloting Planes is possible using communication between APSs. In this case, APSs from two or more administrative domains negotiate their federation. The aspects of federation, negotiation and governance will be treated in detail in Sections 4.2.1 to 4.2.3. A policy-based management system abstracts the be-

haviour and dynamic decisions of a system from its functionality. Therefore, the functionality of a system may stay the same, but its behaviour, specifically with respect to changing contextual conditions, may be varied. Policies are used in the Piloting Plane as a means to adapt its behaviour with respect to changing contextual conditions, for example, changing business objectives, network resource availability or domain membership. As the piloting plane is concerned with the piloting of distributed management systems, its policies are directly associated with the requirements of those management systems, or PAs in the case of Horizon. Essentially, the behaviour of the components of the Piloting Plane, namely the PAs, is dictated by the capabilities of the distributed PAs. For example, a single PA may be deployed using a APS. The behaviour of this deployment may need to be dictated by the "owner" of the PA who will define the policies with which the PAs should be deployed, and the behaviour of the APSs with respect to any coordination with the deployed PA. Other types of policies the APSs should enforce are those delegated to it by the PA to perform negotiation, federation or distribution on its behalf. For example, a PA may inform its associated APSs that information regarding its resources should not be exposed to other PAs unless some strict authorisation requirements are met. APSs can also control PAs that act directly on each virtual networked element. One example for this organization is the handover process for the application continuity. In this case, the handover decision can be controlled by the PAs, while the definition of the parameters for the handover decision, i.e. the maximum number of clients to be accepted on each virtual network, the authentication process being employed, is defined by the high-level goals dictated by the APS.

This organization is shown in Figure 4.1. The high-level APS controls the lower-level PAs setting the policies and SLAs for those components. The PAs act on the virtual resources using interfaces, deploying services and executing FCAPS functions of the virtual resources and nodes. Finally, the lower-level PAs control the near real-time decisions required for the operation of control protocols. As defined before, the Horizon architecture has several levels of policies, mapped into the policy continuous. The piloting plane uses Piloting Policies, which are high-level policies that control the deployment and federation of autonomic control loops. In a sense, Piloting High-level Policies are policies that act upon the inter-domain aspects of the autonomic management domains, controlling when and how two APSs may federate. They may also dictate how the process to resolve conflicts will be carried out by the APs, as well as the deployment and distribution of the PAs in the network. Meanwhile, the APS high-level policies will focus on the management of a single domain. The APSs receive the Piloting High-level Policies as well as the APS High-level Policies from their corresponding administrative par-

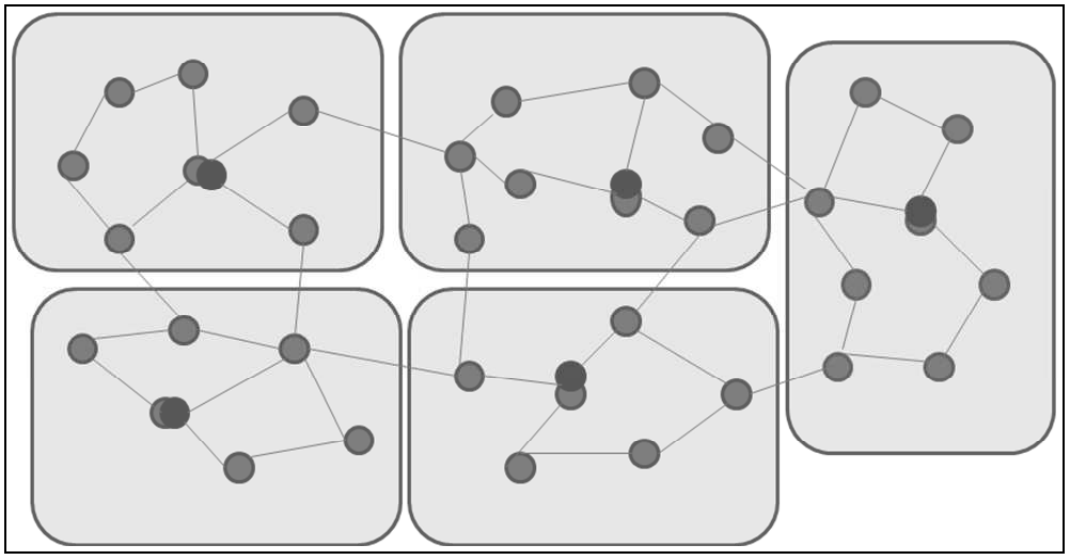

Figure 4.1: Deployment of the APSs in the Horizon architecture.

ties. Those policies are then processed and relevant information from this processing is stored in the System View, and are accessed to the required components as requested. For example, the policies required for the federation of two APSs will be requested from the System View by a Federation Behaviour, while the policies related to security would be requested by the self-security autonomic APS Behaviour. The design of the APS is outlined in Figure 4.2. It enables all components in the system managed by the Piloting Plane to have plug-and-play behaviour. APSs comprise three types of functional elements (FE):

Dynamic Planner FE: acts as a workflow engine for execution of the Behaviours in a APS. This process is necessary to decide what behaviours have to be achieved. The action of the Dynamic Planner will be dictated by policies.

Behaviours: perform the specific/individual piloting actions required to be performed by a APSs and also represent specific management tasks on the network. The main behaviours are Distribution, Federation, and Negotiation. Those Behaviours are described in detail in the following sections. Behaviours act as stubs for the PAs. They also perform internal functions specific to the APSs (core Behaviours), i.e. negotiation among Behaviours.

Situated View FE: is the "local window" of APSs into the Knowledge

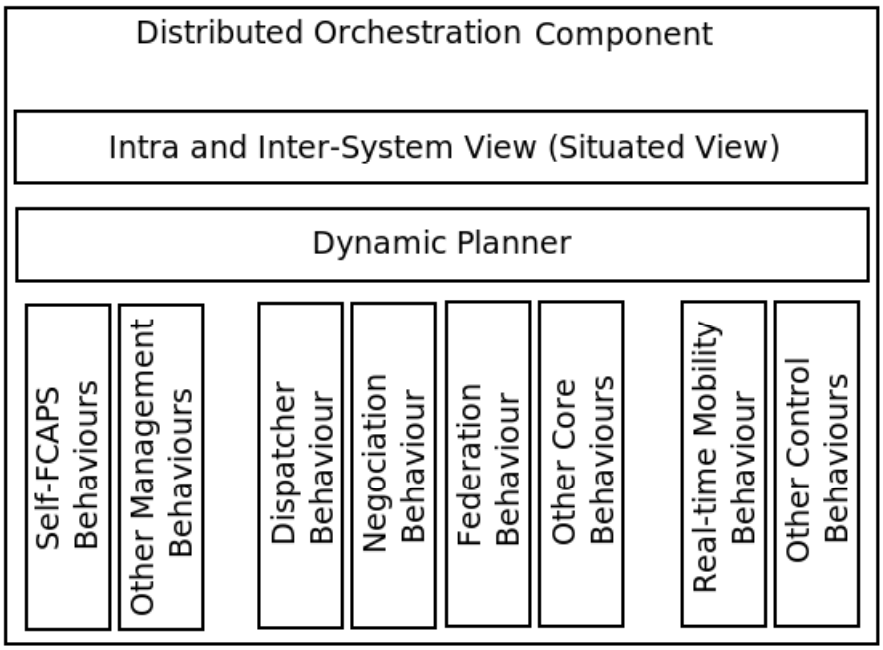

Figure 4.2: Design of the Autonomic Piloting System.

Plane. This view is realized by the knowledge plane and has two parts. The Intra-System View provides an overall, composite view of the system as seen by the components within a single APS. The Inter-System View provides an overall, composite view of the APSs as a whole, that is, it provides a view of the entire Piloting Plane. The respective roles of these components are outlined in more details below.

4.1 Dynamic Planner

The Dynamic Planner acts as a workflow engine for the execution of the Behaviours. Such control includes the following tasks:

- Define the sequence and conditions on which the Behaviours must be bootstrapped for activating a PA (a deployment of the PA components defining their dependencies on other Behaviours; how and where to bootstrap them). It acts as control workflow for all PAs ensuring bootstrapping, initialisation, contextualisation, closing down of PAs. It also controls the sequence and conditions in which one PA invokes other PAs in order to realize some useful functions (i.e., a piloting is the pattern of interactions between PAs).

- Monitor Behaviours, checking that the operation of an individual Behaviour does not conflict with others.
- Forward the advertised high-level policies for each piloted PA and core Behaviour, which were provided by the operator or the owner of the network. Those policies are not processed by the APS, since it is not its function to implement policy refinement.
- Identify conflicts in the initialization and reconfiguration of Behaviours. The Dynamic Planner will then start negotiation Behaviours to reconfigure the deployment plan.
- Trigger the initialization and closing down of Behaviours with a plug-and-play, unplug-and-play approach
- Trigger the dynamic reconfiguration of Behaviours.
- Facilitate the interaction between Behaviours. As an example of interaction, the negotiation Behaviour uses the Dynamic Planner to communicate with other Behaviours in order to solve configuration conflicts. Then, it is up to the Dynamic Planner to trigger a reconfiguration Behaviour.

The Dynamic Planner will be configured by policies, which will help the DP select the Behaviours (and hence which service and/or PA) that must be bootstrapped, the service or PA's parameters and SLAs.

The autonomic control loop of the Dynamic Planner (see Figure 4.3) is realized by the use of policies. Those policies define rules based on events and conditions that must be met by the virtual resources. Whenever those conditions are met, the Dynamic Planner executes the actions defined by the policy, which comprises the bootstrapping of one or more Behaviours. Those policies can be changed on the fly, allowing the DP to reconfigure itself and to adapt to changes in the SLAs of the network. Those policies are called in the Horizon architecture high-level piloting policies. More specifically, the DP will use the distribution policies part of the piloting policies to identify where to instantiate or migrate PAs. Policies will also be used to decide which PA or core Behaviour should be bootstrapped and when. The APSs will also use policies to enforce SLAs and goals into the PAs.

The Dynamic Planner also acts as a mediator for the federation of the PAs. Whenever the Dynamic Planner enforces a new configuration on the PAs, those may refuse it due to their self-governance. In this situation, the PAs will signal this rejection using the appropriate interfaces, and the DP

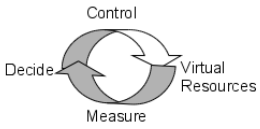

Figure 4.3: General autonomic control loop of the dynamic planner.

will bootstrap a core Behaviour, responsible for conflict resolution, which will propose an alternative configuration.

4.2 Behaviours

Behaviours are sub-components of the APS, which implement a certain piloting task. As an example, the APSs could implement Behaviours for the negotiation of high-level policies, the distribution of tasks, the creation and destruction of services and virtual routers. Behaviours interact with each other when necessary, i.e. the federation Behaviour may interact with a QoS Behaviour if the required QoS couldn't be met when two networks are joined. The lifecycle of a Behaviour is controlled by the Dynamic Planner, which identifies, starts and stops the Behaviours necessary to accomplish a certain piloting task. We tentatively distinguish two types of Behaviours functional elements (FEs): the core Behaviour FEs - required for the proper operation of the Piloting Plane and the PA management Behaviours FEs, related to the control and management of the data plane. The core Behaviour supports the operation of the dynamic planner, implementing the tasks necessary for the proper cooperation of PAs and core Behaviours. Some examples of core Behaviours are listed below:

Distribution Behaviour FE: sends and receives required data to different self-management piloting Behaviour FEs.

Negotiation Behaviour FE: this FE implements algorithms to mediate between the PAs in a APSs so that the PAs can agree on common goals. After the negotiation is complete the PAs should take autonomous and local actions under their corresponding administrative domain, converging to these negotiated goals. Negotiation can also occur between different PAs.

Piloting knowledge update Behaviour FE: manages the dissemination of knowledge regarding the Piloting Plane. It controls information regarding the core Behaviours required for the operation of the Piloting Plane, as well as the information required by the Dynamic Planner.

Network federation Behaviour FE: this Behaviour controls the union and separation of virtual networks controlled by different PAs. This Behaviour identifies the steps necessary to compose/decompose different federated domains, proposing actions to the Dynamic Planner.

Knowledge update Behaviours FE: this class of Behaviours supervises the operation of the Knowledge Plane. They define the "What, When and Where" of the information: What information to collect, when to collect, and from whom (where). Those Behaviours are specific to each service, however the whole set of these Behaviours supervises storage of information in the Knowledge Plane. Each PA requires pre-defined knowledge as well as runtime data. Mapping logic enables the data, represented in a standardized information model, to be transformed into knowledge and combined with knowledge represented by ontologies.

Bootstrap and Initialise Behaviour FE: this Behaviour is capable to bootstrap and initialise other Behaviours under supervision of the Dynamic Planner.

Reconfiguration Behaviour FE: this Behaviour is capable of dynamically reconfiguring and adapting other Behaviours under supervision of the Dynamic Planner.

Optimiser Behaviour FE: this Behaviour is capable of dynamically optimising and organising other Behaviours under supervision of the Dynamic Planner.

Closing down Behaviour FE: this Behaviour is capable of dynamically closing down other Behaviours under supervision of the Dynamic Planner.

Examples of Behaviour FEs, which have direct interworking with other management functions providing near real time reaction are listed below. Those Behaviours will act as a proxy to the PAs, which will then implement each of those functions listed below. The associated Behaviours will deal with the interworking of those functions with others.

- Supervision of service life-cycle managers;
- Supervision of Distribution/Federation/Negotiation of APSs;
- Supervision of interactions between APSs;

- Negotiation / Distribution of the high-level goals to different APSs;
- Monitoring of the APSs; and
- Supervision of network consistency/integrity checks of the sequence of changes to networks made by separate APSs.

In the following sub-sections we provide a detailed description of the main core management Behaviours, which support federation, negotiation, governance and distribution of the management tasks. Next, we describe in more details the PA Behaviours.

4.2.1 Federation Core Behaviour

Each APS is responsible for its own set of virtual resources and services that it governs as a domain. Federation enables a set of domains to be combined into a larger domain, as well as breaking down a domain into smaller domains. With regards to the service enabler, federation Behaviour is assumed to be important when conflicts between services are detected and services can't find a resolution themselves. Therefore, the Federation Behaviour is the only way to find a solution. We are considering two different networks A and B. The APS governs each network as a domain, which can have different requirements, policies and constraints. Federation is set up into three different possibilities: The first case is that APS can federate the network by creating a domain coming from the union of both networks that are in conflict. The resulting domain inherits the characteristics of their union. In the second case, we can assume that APS can federate by creating a domain resulting from the intersection of both networks. In this case, the new network's requirements are the common ones from each network. The third case, we might assume that APS can't federate by the two previous possibilities. Moreover, in this case, the APSs create a bridge, enabling to link the networks.

Here are two aspects to federation in Horizon. The first deals with the agreements that must be shared among all participating members of the federations, and the second deals with facilitating the provision of services across federations. These two aspects are treated separately, but are tightly dependent on each other because if agreements cannot be made between participating members of the federation, then there can be no consensus on service provision. One of the major challenges facing the Internet of today is that dynamic agreement adaptation between independent administrative domains is difficult when business and technical concerns need to be addressed

individually. In this case, the APSs will start up a special negotiation Behaviour to help in the negotiation of a new agreement, which will be driven by the decisions of the APSs. In Horizon, federation of APSs is separated into two concerns, that are tightly coupled, namely business concerns and technical concerns. These are summarised hereafter:

Federation of High Level Objectives

A federation in Horizon is born when two or more APSs need to come together under a common objective. Typically, this objective is to provide a common set of services with a guaranteed reliability across the boundaries of the APSs. Each APS will contain its own business objectives as defined by the policy continuum. They need to decide on a common set of business objectives that can be maintained across the federation. Once these set of common objectives are put in place, the business aspect of the federation is addressed. However, modifications to the federation business objectives can be proposed by any member, and any member can choose to leave the federation. Leaving a federation is part of the self-governance part of the PAs and may entail a penalty if it breaches the terms of the federation. As each APS can decide for itself to participate with other APSs, the federation merely puts in place some common understanding between federation members about how they should interact with each other. One example could be to collude against a specific APS that breaches the terms of the federation. Actually offering and consuming services from within a federation is a technical concern and must be dealt with separately. This is to ensure that new services can always be introduced into a federation.

Technical concerns of a Federation

Once an APS is participating in a federation of APSs, it can consume and provide resources and services within the federation. The APSs in Horizon is concerned with the actual piloting of the usage of these services and resources, and the joining and leaving operations of a federation. It is the challenge of the APSs to ensure that requested services of the APS are made available in a way that abides by the terms of the SLAs and policies specified on the federations. To do this, the APSs must be able to assess whether the configurations of services are adequate enough to ensure the agreed terms of the federation are being upheld. The PA APS must also abide by its own business objectives and if these objectives are no longer fulfilled it may decide to leave the federation. On leaving a federation, the associated APSs must signal this intention to the federation members.

Different types of services and resources required different technical solutions to ensure that they can be used effectively in a federation. It is up to the service creator to ensure that if the new service is to be used within a federation, that it be technically feasible to do so.

4.2.2 Distribution Core Behaviour

The APSs provides communication and control services that enable tasks to be split into parts that run concurrently on multiple APSs within an Piloting Plane, or even across multiple Piloting Planes.

Business concerns of Distribution

The business concerns relate to whether each APS requires the distribution of information, tasks, and code. Information distribution is important, as some APSs may generate information that is unique to them and will need to be explicitly controlled. Access control policies may play a role here. Task distribution can be seen as the distribution of work that involves distributed processing across a number of processing elements. For example, the computation of the effective bandwidth of a particular film may be distributed across a number of machines, as this may be a time consuming process. The business concern here is that a particular APS may request its associated PAs to distribute a processing task. The distribution of code is slightly different from the distribution of information and tasks in that there is typically no return expected from the distributing PA. The distribution of code may be an enabler for an APS to upgrade a particular service offering that relies on the federation of a number of PAs.

Technical concerns of Distribution

The distribution of information may be carried out using the concepts being developed in the knowledge plane relating to the Context Information Service. However, the APSs need to be involved as it can enforce strict access control over the information, and can thus instruct the distribution of information to abide by the high-level policies of the APS. The high-level policies may authorise or prohibit information from leaving or entering its system. The distribution of tasks can be technically carried out by the platform being used to distribute information. However, the task information may be in a specific format, where strict instructions may need to be in place to instruct the target PAs of the requirements of the task. It is up to each PA to decide whether to accept or reject the task being distributed. The PA will need to inform the APSs of its intentions; the APS then takes care of the distribution, and informs the PA whether or not the intentions were met.

The distribution of code can also be carried out by the platform being used to distribute information. Code in this case is a special form of information that can be distributed, in that it is mark to be executable. The code may be used to upgrade or deploy a new service within other PAs. Therefore, the PA instructs the APSs that code needs to be distributed and it is up to the APSs to handle the distribution of the code. This can be carried out using the Service Enabler Plane.

4.2.3 Negotiation Core Behaviour

In Future Internet Networks approaches, it becomes mandatory for network and service providers to offer and publish their services so that more complex services can be provided. An important component that allows this requirement to be met in Horizon is the negotiation component part of the Piloting Plane. This component acts as a virtual service broker that mediates between different PAs, taking care of service requests and providing support so that the underlying service providers can negotiate responsibilities, tasks, high-level goals, etc. This support should be carried out taking into account the nature of the underlying service providers, the services they provide, their interests, their service qualities and other key aspects. All in all, this functionality should be provided leveraging the service requester entities from complex decision-making processes.

In Horizon each APS advertises a set of capabilities (i.e., services and/or resources) that it offers for use in the Piloting Plane. The negotiation component enables the specific functionality of selected capabilities to be agreed upon between APSs. Examples include using a particular capability from a range of capabilities (e.g., a particular encryption strength when multiple strengths are offered), being granted exclusive use of a particular service when multiple APSs are competing for the sole use of that service, and agreeing on a particular protocol, resource, and/or service to use. The negotiation functionality piloted between APSs and PAs has inherent business and technical concerns. The following elaborates on these two critical aspects.

Business and Technical Concerns for Negotiation

When the PAs negotiate high-level goals under an APS, or when different PAs negotiate high-level goals with other PAs, the negotiation finishes when the participants align their internal business objectives with a common one, namely when the participants converge to negotiated high-level goals for which virtual and non-virtual resources must be allocated, managed and controlled autonomously in their respective domains (PA and/or APS). In the negotiation of business objectives, the responsibilities, benefits and penalties are also considered during the negotiation.

It is worth mentioning that an APS and/or a PAs may negotiate business objectives with several PAs and APSs, sequentially or in parallel. The negotiation of business objectives is in turn influenced by technical concerns in the sense that APSs compromise resources to fulfil the negotiated high-level goals. The governance capability of an APS (and that of a PA) defines with whom, why, and when to negotiate. The negotiation capability ensures that APSs and PAs can always negotiate with other entities in a federation.

As PAs and APSs can have active negotiated agreements with several

parties, there is a potential need to re-negotiate the high-level goals due to statistical changes in the resources committed to these goals when they cannot be fulfilled, or due to some internal decisions that lead to such corrective actions.

The APSs may also trigger re-negotiation when the common business objectives are not fulfilled. Re-negotiation of high-level goals is a functionality that should be supported by the Piloting Plane. It is part of the governance capability of each PA and/or APSs to decide on which of its negotiated business objectives to re-negotiate. The renegotiation can be driven by utility functions, cost/benefit optimisations, etc. Again, during the re-negotiation of high-level goals, the responsibilities, benefits and penalties are also considered.

The Piloting Plane provides the means for the mediation between PAs and APSs so that they can negotiate high-level goals as a result of a complex service request. As the PAs and APSs may belong to different administration domains, they may talk different languages, may express their high-level goals in different terms and so forth. Ontology translation and mapping techniques can be used to create the common language upon participating entities can negotiate, federate, etc. The Piloting Plane must provide the mechanisms for the negotiation to occur regardless of any of these technical aspects.

4.2.4 Piloting Core Behaviour

The governance functionality of an APS deals with the self-interested actions that it takes to (re-) negotiate high-level goals with other parties and to take actions that can involve the commitment of virtual and non-virtual resources that may help provision of services across federations. Following on, each PA can operate in an individual, distributed, or collaborative mode. In each case, it collects appropriate monitoring data in order to determine if the virtual and non-virtual resources and services that it governs need to be (re-) configured. Business objectives, service requirements, context, capabilities and constraints are all considered as part of the self-interested decision making process. APSs also may be federated with other APSs (Inter-system situated view) and as such they may act self-interestedly, namely they should have self-governance properties.

Business and Technical Concerns for Governance

In a federated environment, the APSs provide support so that their underlying PAs are aware of the needs of other PAs within a federation. As an APS is aware of its needs, it can decide which set of other PAs it collaborates with. The nature and scope of the business objectives of an APS would result in collaborations that can be different on the types of functionality and

services that are the subject of the collaboration. The APSs would provide support for these self-interested decisions to take place and to be acknowledged by the participants during the negotiation phase within a federation. As introduced earlier, each APS can operate in an individual, distributed, or collaborative mode:

- The APSs act as individual entities when they are not part of any federation, when they work isolated and autonomously, governing its own virtual and non-virtual resources and services driven by its own business objectives. No common goals are shared or responsibilities from any APSs are acquired. The APSs should support this functionality by handling and transmitting the messages sent by the corresponding PA to other PAs and/or APSs in the federation they are located in.

- As a result of the distribution function of the Piloting Plane, APSs can work with other APSs in a federation where complex services are provisioned by coordinating the activities, resources and services of each distributed APSs after a negotiation phase. Relevant to the governance function is the fact that each APS should provide a number of contract interfaces to the PAs, which use them to promote and mediate the negotiation of a complex service, its activation and maintenance. These interfaces could be interpreted as contract interfaces that isolate the internal structure and capabilities of the PAs.

- An APS works in a collaborative mode when it acts as a local or a global collaborator. A local collaborator is responsible for coordinating the functionality of other APSs in a given Piloting Plane, this is when the APSs delegates part of its control to an APS. An APS works as a global collaborator when it coordinates the functionality of APSs across different Piloting Planes, namely amongst APSs. This enables the emulation of client-server, n-tier, clustered, and peer-to-peer architectures.

In any case of operation, the PP should provide the means for an APS to govern its virtual and non-virtual resources through a well-defined set of interfaces. The overall aim of the APSs with this regard is to allow the PAs to always decide on the action to take based on self-interested policies inside the APS, driven by its business objectives and capabilities. Its decisions should also take into account the policies of other APSs participating in the federation (see Sections 4.2.1 and 4.2.3). However, the APS can reject to abide to certain policies of the federation when those are not aligned with its own business objectives. In this case, it is up to the APSs to re-negotiate or change the policies of the federation.

4.2.5 APS Behaviours

The APSs use the APS Behaviours as an interface to communicate with the PAs. This encapsulation allows the APSs to see PAs in a uniform way, having the same interfaces as the core Behaviours. All the communication of the PAs with the APSs follows the information models defined in WP3. Further, APS Behaviours are wrappers to the CPA, similar to stubs and skeletons in RPC (Remote Procedure Calls) or CORBA, once they provide a translation from the specific design issues of a APS to the operational philosophy of the APSs. One of the uses of such a wrapper is to hide from the APSs the different implementations of the APSs, for example ensuring a single communication point, even though the APS may be a distributed component spread around several virtual nodes.

The wrapper also defines a set of commands that the APSs must support to allow the APSs to orchestrate their federation and distribution. The interface provides mechanisms for the APs to disseminate and renegotiate policies to the APSs, allowing the PA to be self-governing. The PAs can also support near real-time control of virtual resources and protocols. While the high-level self-FCAPS APSs are mainly concerned with long-term management of resources, lower-level PAs are deployed to react as fast as possible to changes in the PA-aware context. Thus, lower-level PAs use simple algorithms that act based on a pre-determined overall goal. Those goals are dictated by policies defined by the APS. Those PAs will act over a single virtual resource or a small set of virtual resources, due to the time constraints on their reaction. In a sense, lower-level PAs may be seen as the first control loop of an autonomic system, acting based on pre-determined goals and with no sort of embedded learning. Examples of possible technologies that could be employed are state machines, PID controllers, fuzzy decisions, etc.

Low-level PAs are associated with the maintenance of a QoS defined by the network SLA, controlling the parameters of the virtual nodes as well as their running algorithms, for example mobility management algorithms, the queuing discipline of QoS-aware MAC protocols or the admission and authentication of applications.

4.3 Intra- and Inter- system views

APSs use the knowledge plane to store and disseminate the information required for their operation. The information can be decoupled into two parts, or views, according to their relevance to a given APS.

The Intra-System View concerns information required to orchestrate the

services within the piloting domain, while the Inter-System View deals with the piloting of several piloting domains. The Intra-System View contains information that enables APSs to become aware of the particular situation that they are now in; the Inter-System View provides similar information for collaborating APSs.

The Intra-System View thus deals with the services being run within the boundaries of an APS, together with the information relevant for the operation of this APS. This view is also called the Situated View. There is an important trade-off on the definition of the Situated View. Larger Situated Views will allow decisions to be taken using more knowledge describing the overall sate of the system. However, the cost of disseminating the knowledge among all the concerned nodes will be higher, as well as the processing cost. As a consequence, an optimal Situated View could be defined based on the problem being solved and on clear performance metrics.

The situated view will be detailed in Deliverable D3.3, once it is a part of the Knowledge Plane. It will also employ the ontologies and information models of D3.1 in the representation of the knowledge. The APSs maintain the situated view, defining what information must be stored in this view, from which virtual nodes or resources, and with what frequency it must be updated. The interface of the Piloting plane with the Knowledge Plane is described in Section 4.4 below.

4.4 Interfaces of the APS

This section describes the interactions of the APSs with other elements of the Horizon architecture, as shown in Figure 4.4. We first describe the interface of the APSs with the Knowledge Plane. The interfaces with virtual resources, the APSs and the service enablers plane are just introduced here, since they will be defined in the deliverables respective to each of those functions.

The knowledge base consists of a warehouse where all information and knowledge required for management and control tasks is kept. As each element within the autonomic architecture should be able to work autonomously, the Piloting Plane should also be able to provide autonomously the information required for its functionality. The piloting plane provides the APSs with the required parameters regarding to which information should be monitored, how often it should take place and from where in the network it should be gathered. These parameters are decided by Dynamic Planner, based on the functionality of Piloting plane and the state of the network, and raise a specific behaviour dedicated to this task. So, we can consider two types of

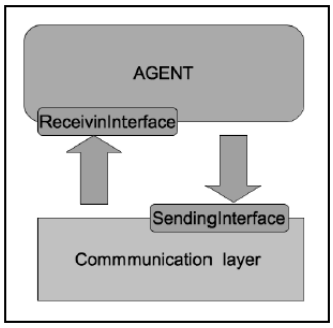

Figure 4.4: Interfaces of the APS.

knowledge within the knowledge base, those related to Management tasks and those related to Piloting tasks.

One of the objectives of the PP is to identify what is the needed information that is required for autonomic decisions, and next to activate the required KP functions that will ensure the timely collection and delivery of this information to the APSs or to a specific Behaviour. The format of the information requests, as well as the information that is produced by the PP follows the information models being produced in the Work Package 3 of the project. The following functions are required for the KP/OP interaction:

Lookup(Info): requests the lookup of a certain piece of information (or knowledge) to the knowledge plane. This is used for fetching policies, SLAs as well as context and relevant configuration parameters of the APSs being piloted.

Store(Info): stores a certain information on the knowledge plane. This function is used for knowledge produced within the OP, which is then stored in the KP.

Subscribe(Event, Component): defined by a condition on the stored information of the KP, this function allows the APSs to be notified of relevant events happening in the network. The events may define a condition and also a set of nodes or APSs where such an event may happen. For example, the APSs may be interested in watching for the occurrence of certain faults to trigger the reconfiguration of an APS responsible for fault management. The subscription also identifies which APSs and which of its components will process the information, that is, it can be the Dynamic Planner that requests the information, or it may be necessary for the operation of a certain Behaviour.

Watch(Info) and unWatch(Info): used for the APSs to define which are the information that must be periodically collected and disseminated by the KP. For example, the averaged used bandwidth of a network interface associated to a certain PA may be monitored to verify if this component should be migrated to another node on the network. The information fetch is defined by the type of the information, its situated view (from which virtual nodes it must be collected), the periodicity of such an update and the component requiring this information. Once the APSs issued a Watch request, the KP is responsible for the maintenance of the information, delivering it directly to the concerned component. One example of use of Watch would be a APSs that defines that the number of active flows on the network should be communicated every 30 seconds to the autonomic performance optimization APS. When this information is not needed anymore, the APSs will use the unWatch() function to free the KP resources.

Push(Information, PAs): used for synchronous message passing among the PAs. The Subscribe, Watch and Push interfaces of the APSs are used by the APSs to create their inter-system view, used to feed the Dynamic Planner and the core Behaviours with their correct parameters. Those functions are also used for inter-APS communication, once a APSs may watch the state of another APSs using the same functions. The access to information from other APSs should be controlled by access policies, once APSs may be controlled by different organizations, and thus it may be in the interest of the APSs to hide some aspects of the management of the network, e.g. due to competitive issues or to limit the knowledge of the topology and operation of the network to ward security attacks.

Interface SEP/APS, Virtual Resources/APS and PA/APS: As described before, the ultimate task of the piloting plane is to deploy PAs within the network. The deployment of PAs should be done considering the management requirements of services. Consequently, the piloting plane performs this task of PAs deployment, using the Service Enabler Plane (SEP). However, the PP uses the SEP to deploy PAs, it does not deal with the deployment of services. In one hand, the piloting plane needs to interact with virtual resources in order to provide the PP with physical management and control information regarding the state of the network and resources, which will be used for deploying PAs Behaviours. This interface is useful since it provides the necessary information helping to deploy PAs in the adequate places regarding

the state of the network and resources. The last interface used by the APS, which has been described before, is the interface of the APSs with the PAs. This interface is indeed implemented by the PA wrapper Behaviours, as described in Section 4.2 of this deliverable.

Chapter 5

The Piloting Agents

We describe in this section the distributed Piloting Agent plane. This plane is composed of Distributed Intelligent Agents each associated with a Network Element (NE) as illustrated in Figure 5.1.

By distributing agents across the home network, the network piloting plane permits to deal locally with local problems. Indeed local problems are often simpler and easier to deal with than the resulting global problem. Furthermore a local problem can be addressed earlier locally than in a centralized approach; e.g. an agent can immediately change the configuration of its network element to react to a local load problem. Beyond purely local problems, agents cooperate among neighbours to deal with problems appearing in the neighbourhood, e.g. a connectivity problem can be detected by several agents. These agents can then cooperate to characterize the problem more precisely and eventually provide a solution or a synthetic report to the global hypervisor. The Piloting plane draws from well established, simple and powerful Distributed AI techniques to program Distributed Intelligent Agents individually and their interactions. The knowledge plane which is

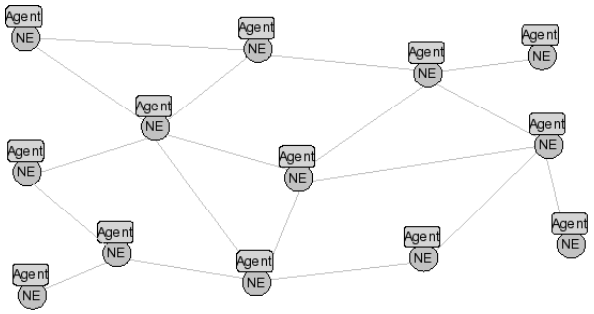

Figure 5.1: Intelligent agents forming the Piloting Plane.

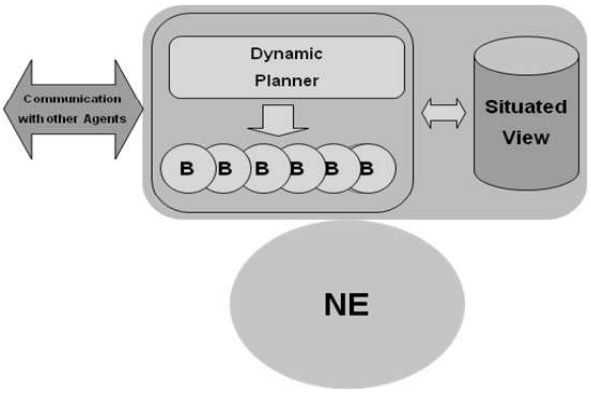

Figure 5.2: Outline of Agents Architecture.

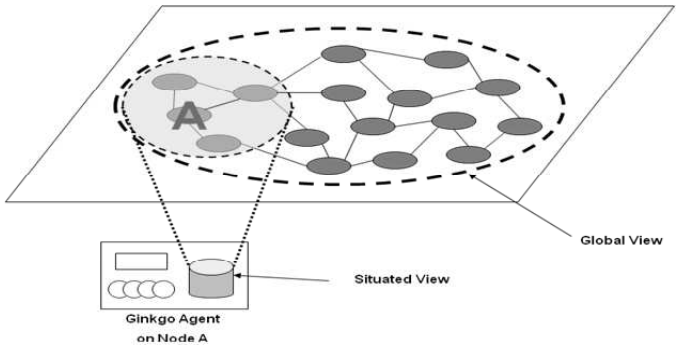

Figure 5.3: Each Agent has its own Situated View of the Network.

a part of the Piloting plane consists in the situated view component of the agent which is an information base of gathered information. Similarly, the orchestration plane is composed through a Dynamic Planner and different Behaviours. The architecture of the piloting system is outlined in Figure 5.2.

The first part of the piloting plane is composed of a Knowledge plane which is a distributed information base embedded in each agent. It contains local information of the agent as well as global information of the network. Each agent maintains its own view of the home network on the basis of information obtained (i) directly from local observation of its network element (NE), (ii) indirectly for the rest of the network by exchanging information with its neighbours. This agent-centric view of the network, focused on the agent's close network environment, is called the Situated View, and is illustrated in Figure 5.3.

The rationale for the Situated View is that events occurring in the neighbourhood of an Agent are generally of greater importance for the agent than events occurring in a remote part of the network. The fact that local events

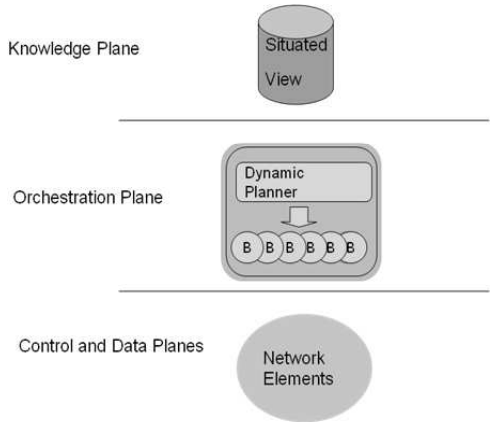

Figure 5.4: Outline of the Piloting system.

are known earlier and are more accurately documented in the Situated View makes it easier for the agent to react rapidly and appropriately. Agents regularly check for important changes appearing in their Situated View - and thus in the network environment as seen by each agent - and may decide to automatically adapt certain parameters of their own NE or ask neighbouring Agents to do so for their respective NEs. The use of the Situated View drives implicit cooperation between agents who "influence" each other via the knowledge that they are sharing. Implicit cooperation is the primary mode of cooperation among agents in the Piloting plane. This mode of cooperation is simple, particularly robust and well suited for dynamically changing environments because it does not require the establishment of an explicit dialog and a strict synchronization between Agents.

The role of the part named orchestration plane is to decide, in real time if necessary, how to feed algorithms, such as mobility, security, QoS, etc. An autonomic system is composed of different autonomic elements, which cooperate in order to achieve the overall objective of the autonomic system. Therefore, an orchestration plane is needed to decide in real time what to do when a threshold is reached. The autonomic architecture performs this process via behaviours and dynamic planner. What an agent is capable to do is defined as a set of Behaviours ("B" in Figure 5.4). Each of these Behaviours can be considered as a specialized function with some expert capabilities, able to deal with specific aspects of the work to be performed by the agent.

Typical categories of Behaviours are as follows:

- Updating the Situated View in cooperation with other agents;
- Reasoning individually or collectively to evaluate the situation and decide to apply an appropriate action, e.g. a Behaviour can simply be

in charge of computing bandwidth availability on the NE, it can also regularly perform a complex diagnostic scenario or it can be dedicated to automatic recognition of specific network conditions and

- Acting onto the NE parameters, e.g. a Behaviour can tune QoS parameters in a DiffServ context.

Behaviours can access the Situated View which operates within each agent as a whiteboard shared among the agent's Behaviours. The activation, dynamic parameterization and scheduling of Behaviours within an agent is performed by the Dynamic Planner. The Dynamic Planner decides which Behaviours have to be active, when they have to be active and with which parameters. The Dynamic Planner detects changes in the Situated View and occurrence of external/internal events; from there, it orchestrates the reaction of the agents to changes in the home network environment.

Chapter 6

Testbed

It is important to evaluate the proposed self-management system before deploying the system in a real network. Figure 6.1 illustrates the testbed built with this objective.

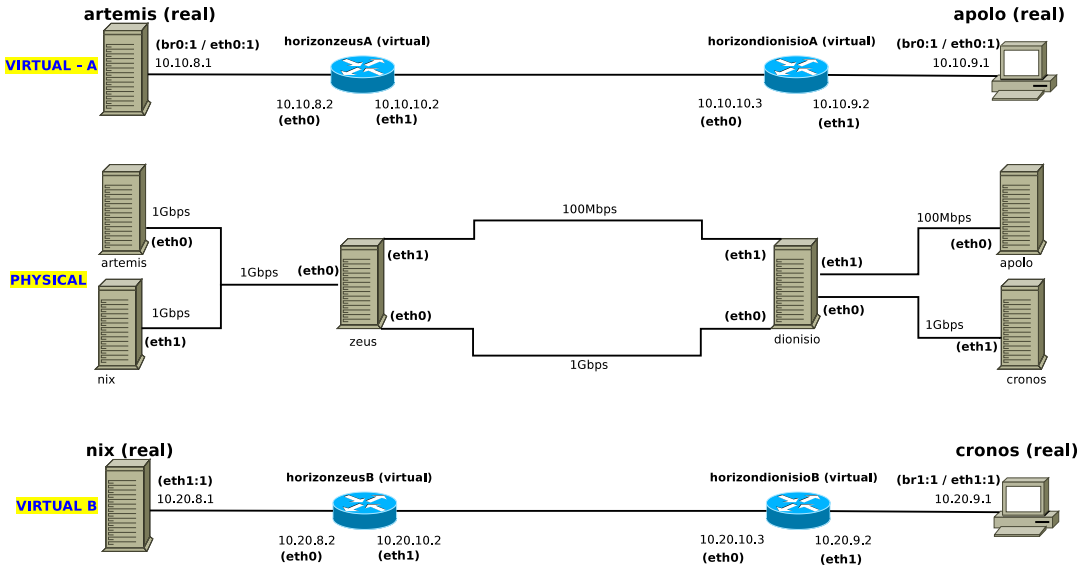

Figure 6.1: Testbed.

The testbed contains two virtual networks (virtual network A and virtual network B). Both virtual networks contain two virtual routers each. The virtual routers are located at the real hosts *zeus* and *dionisio*. To the virtual network A be instantiated, it is needed to instantiate the virtual routers *horizonzeusA*, at the real host *zeus*, and *horizondionisioA*, at the real host *dionisio*. Similar instantiations are needed to the virtual network B.

Virtual network A is created to interconnect hosts *artemis* and *apolo* through a 2-hop virtual path. Similarly, virtual network B is created to interconnect hosts *nix* and *cronos*. The 2-hop paths of each virtual network can be mapped on one of two possible physical paths between the real hosts *zeus* and *dionisio*: an 100Mbps link and an 1Gbps link. Initially the two virtual paths share the 100Mbps link.

Preliminary experiments were performed to confirm the possibility to change the mapping of the virtual paths during the operation of both virtual networks. The idea was to overload the 100Mbps link and, after that, to migrate just one virtual path to the 1Gbps link. This experiment simulates a scenario where agents located at the routers detect the high utilization of a link and take the decision to migrate one of the virtual links.

The next sections present more details about the testbed. Section 6.1 summarizes some tools used to build the testbed and Section 6.2 presents the results of the preliminary experiments. It is important to highlight that these experiments were realized manually without mediation of agents from the proposed self-management system. The results obtained with the agents are presented in Chapter 8.

6.1 Tools

In this section we describe the major tools employed to build the testbed used in the experiments. The tools are used for the deployment and manipulation of virtual networks, and the Ginkgo platform was used to the development of the multi-agent system.

6.1.1 `qemu`

`qemu` [24] is a processor emulator which can also be used as a virtualization platform. Comparing to other virtualization platforms, `qemu` has the advantage of being easy to install, because to the host operation system it is an application like any other. The disadvantage is that `qemu` is susceptible to the same process scheduling and memory management algorithms of the host operating system.

`qemu` is a free software under the GPL and open-source. There are some extra components which can be used with `qemu` to improve the performance of the guest operating system when the real processor contains the same instructions of the emulated processor. The idea is to allow the guest operating system to access the real processor directly.

6.1.2 KVM

The Kernel-based Virtual Machine (KVM) is a full virtualization hypervisor based on the machine emulator qemu. It runs on x86 architectures with new virtualization technologies like Intel VT and AMD-V. KVM consists of a kernel module of Linux, which, when loaded, provides an interface at /dev/kvm to the setup and control of the guest virtual machines.

KVM is a free software under the GPL and open-source that allows to use external tools to control it, like libvirt.

Some of KVM features that are interesting for the Horizon project include:

- Good performance in full virtualization.
- Live-migration of virtual machines.
- Support SMP hosts and guests.
- Memory ballooning.
- VM networking by bridging, routing or private networks.

KVM surpasses qemu in terms of performance in full virtualization because the main objective of qemu is to be a processor emulator, not a virtualization platform. However, the installation and configuration of qemu is much easier than KVM, since there is no need to modify the host operation system with special kernels or modules.

The preliminary experiments of the multi-agent system presented in this report were realized in a testbed powered by qemu virtual machines. Now, the testbed is powered by KVM virtual machines. The final experiments of the multi-agent system, presented in Chapter 8, were performed on this current version of the testbed.

6.1.3 libvirt

libvirt is an API to access the virtualization capabilities of Linux with support to a variety of hypervisors, including qemu, KVM and Xen, and some virtualization products for other operating systems. It allows local and remote management of virtual machines. With libvirt it is possible that an agent uses the same code to request information regarding the performance of a virtual link independent of the hypervisor running in the virtual routers.

The libvirt is implemented in C (supporting C++) and includes direct support for Python. It also supports a number of language bindings which

have been implemented for Ruby, Java, Perl, and OCaml. During the work of this report, the C version and the Java binding of the API were used.

The importance of `libvirt` for this work is to make virtualization technology-independent, facilitating future design changes. Thanks to `libvirt` was possible to migrate the virtualization platform in the testbed from `qemu` to `KVM` without modifying the agents.

6.1.4 `Ginkgo Distributed Network Piloting System`

Ginkgo Distributed Network Piloting System [25] is an agent platform based on autonomic networks. It has the building blocks for the development of a piloting system for computer networks. The framework allows the creation of lightweight and portable agents, which facilitates its implementation in heterogeneous environments: routers, switches, hosts, wired and wireless networks. The agents play the role of the autonomic manager of autonomic computing. With distributed managers near its managed elements, monitoring can be done locally.

The platform allows to form clusters of agents in neighborhoods. Neighbors exchange information and get a situated view of the network. Thus, besides the local environment, the agent is aware of other network places. This information is stored in the knowledge base that has an information model to facilitate communication between agents. Other data repository is the policy file, which contains rules of the application. With the environment and application knowledge, the multi-agent system can provide to network the self-knowledge property.

The sensing, cognition and acting of agents are realized by the behaviors. They feed the knowledge base, perceive and predict threatening events and perform changes on the managed elements. Agents also have a dynamic planner that, with information in the knowledge base and the rules in the policy file, changes parameters of the behaviors and controls the life cycle of the agent. This makes possible to develop the properties of self-configuration, self-healing, self-optimizing and self-protection in the network, which promote the self-management.

The Ginkgo Platform provides the key features necessary to implement the Piloting Agents presented in Chapter 5.

6.2 Preliminary Experiments in the Testbed

The preliminary experiments were realized to evaluate if it was possible to change the mapping of the virtual links during the operation of the virtual

networks. This section summarizes the results obtained with one of the experiments.

The virtual routers are VMs based on `qemu` version 0.12.5. For the creation and manipulation of them, it was used the libvirt in version 0.8.3. Both the physical and virtual machines contain the operating system Debian GNU/Linux with kernel version 2.6.32. The virtual links are created in the data link layer, with the Ethernet protocol, via virtual interfaces and bridges. The bridges are controlled by the utility `brctl` of the Bridge-util package, version 1.4-5.

Initially the two virtual networks had their virtual links sharing the 100Mbps link of the substrate network. Traffic was generated using the `iperf` in the two virtual networks until the 100Mbps link was saturated. At this point, scripts were executed manually at the routers of virtual network.

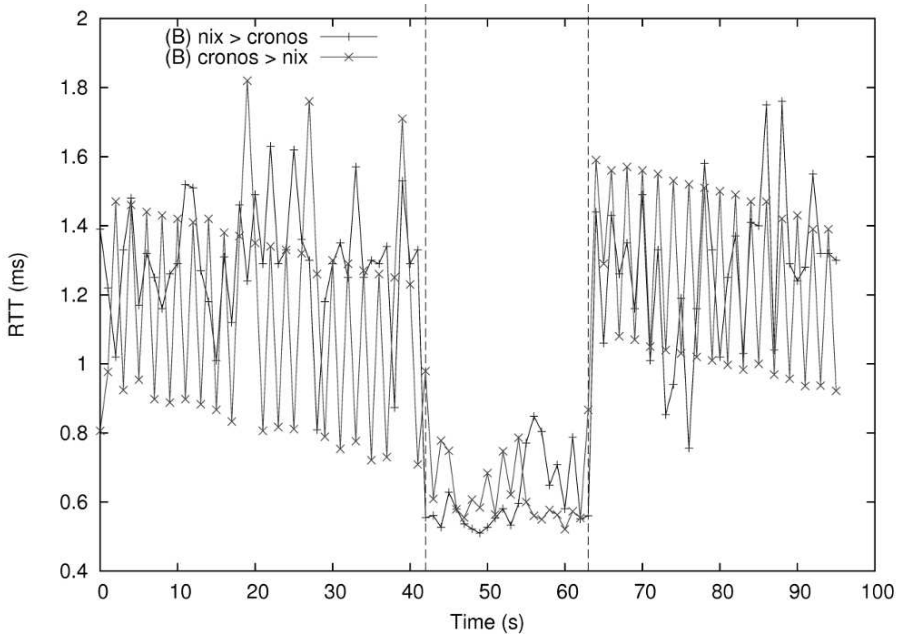

Figure 6.2: RTT between hosts of the virtual network B.

The graph of Figure 6.2 plots the RTT between the two hosts cronos and dionisio when the communications used the virtual network B and the 100Mbps link was saturated. The scripts to change the mapping of the virtual link were executed at time 40s. It is possible to observe in the graph that this change had as consequence the reduction at the RTT between the two hosts. At time 60s, the scripts were executed to return the mapping of virtual link to the 100Mbps link. As consequence, the RTT increased to its initial range of values.

An experiment similar to the one summarized in this section is described in Chapter 8. The difference is the last one was realized with agents taking actions, instead of the manual execution of scripts.

Chapter 7

The Multi-Agent System

We implemented a multi-agent system within the testbed to perform the management of virtual networks. The system was developed with support from Ginkgo platform, version 2.0.13. The agents were compiled and executed with the version 1.6.0-21 of Java. For the monitoring of virtual networks by the agents, the `libvirt-java` library version 0.4.6 was used.

The aim of the experiment is to change the mapping of virtual links over paths in the substrate network dynamically and automatically, according to context changing, instead of manually as presented in Section 6.2. The resources being monitored are in the virtual networks, not in the physical network. It is monitored, for example, the failure of a resource in the data plane, which can lead to faults in the management system.

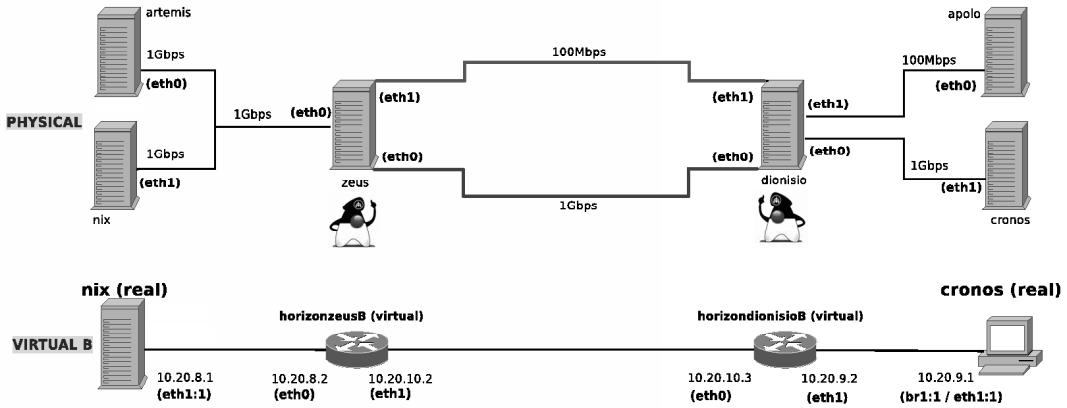

Figure 7.1: Agents in physical routers managing virtual routers.

In the knowledge base, each network resource is an individual of the information model. Each resource is controlled by a single agent and, therefore, each individual is unique in the knowledge base even after the diffusion. In

the information model, the links are directed, i.e., each wire is represented by two links in opposite directions. The agents are located in routers and they send data over a directed link, monitor and control this link. Figure 7.2 shows the class diagram of the information model, based on [26].

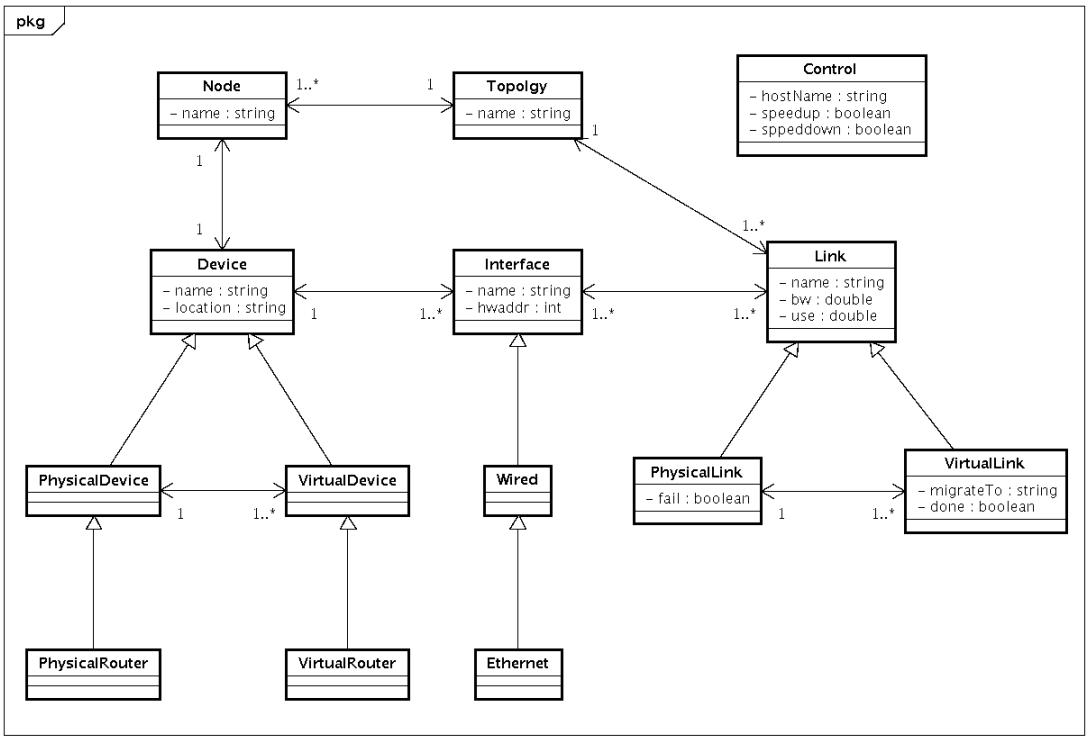

Figure 7.2: The information model of the agents.

The neighborhood consists of all agents of the routers within the domain, so the information is exchanged by broadcast. To make a change in the mapping of the virtual link is necessary to perform actions on both ends of the physical path. Two agents in separate routers are responsible for a part of the execution. Ideally, the two actions occur at the same time to reduce losses in the data plan, so, we must create a mechanism to synchronize agents. All communication on the Ginkgo platform is based on the diffusion of knowledge base, then, we need to extend the information model in order to indicate the status of the resources and the agents. With this information, behaviors are organized to perform the actions at about the same time.

The agents have four behaviors: Monitor, Analyze, Plan and Execute, which are executed periodically in sequence forming the autonomic cycle of the manager. Their actions are listed below.

Monitor: collects data from network interfaces through `libvirt` and feeds the knowledge base.

Analyze: verifies whether the use of physical links exceeded the threshold defined in the policy and inform in the knowledge base which are overloaded.

Plan: if there is a problem with a link belonging to the agent, it plans the action, i.e., it chooses another physical link, if any, to receive the flow of a virtual link that is mapped on the overloaded one. If the problem is on a link in an adjacent node, it plans an action consistent with what was intended by the neighbor agent.

Execute: ensure that both agents have already planned their actions to perform its part locally, states the conclusion of the execution in the knowledge base.

There is an inversely proportional relationship between the consumption of resources for management and speed of the corrections. If the frequency of the agent life-cycle is high it will use more processing cycles of the router and take corrective actions more quickly. If the frequency is low, the processor consumption will be lower and the fixes will take longer. To obtain a good trade off between them, saving resources without losing too much performance, we defined rules in the policy file to reduce the time between cycles when the link usages are above the threshold, and to return to normal rate when the execution ends. The source of policy file is listed in Figure 7.3.

```
(policy (subgoal main (rules
  (rule RINIT if (impulse init) (
    (set control.speed 1.0)
    (changerate MonitorBehavior 1.0)
    (changeprio MonitorBehavior 3)
    (start MonitorBehavior)
    (setcontrol AnalyzeBehavior.threshold 0.5)
    (changerate AnalyzeBehavior 1.0)
    (changeprio AnalyzeBehavior 2)
    (start AnalyzeBehavior)
    (changerate PlanBehavior 1.0)
    (changeprio PlanBehavior 1)
    (start PlanBehavior)
    (changerate ExecuteBehavior 1.0)
    (changeprio ExecuteBehavior 4)
    (start ExecuteBehavior)))
  (rule RFASTER if (and control.speedup (= control.speed 1.0)) (
    (set control.speed 0.2)
    (changerate MonitorBehavior 0.2)
    (changerate AnalyzeBehavior 0.2)
    (changerate PlanBehavior 0.2)
    (changerate ExecuteBehavior 0.2)))
  (rule RNORMAL if (and (not control.speedup) (= control.speed 0.2)) (
    (set control.speed 1.0)
    (changerate MonitorBehavior 1.0)
    (changerate AnalyzeBehavior 1.0)
    (changerate PlanBehavior 1.0)
    (changerate ExecuteBehavior 1.0))))))
```

Figure 7.3: The rules of the policy file.

Chapter 8

Results

In the experiment, the virtual routers are VMs based on `KVM` version 0.12.5. For the creation and manipulation of them, it was used the `libvirt` in version 0.8.3. Both the physical and virtual machines contain the operating system Debian GNU/Linux with kernel version 2.6.32. The virtual links are created in the data link layer, with the Ethernet protocol, via virtual interfaces and bridges. The bridges are controlled by the utility `brctl` of the Bridge-util package, version 1.4-5.

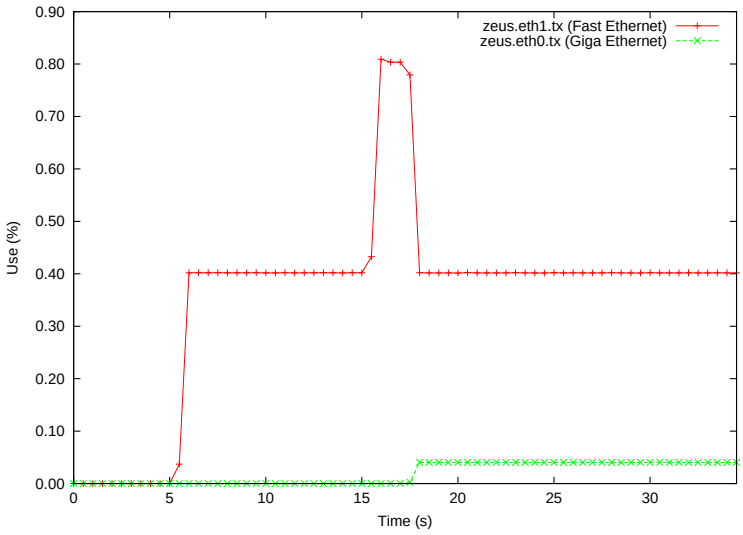

Figure 8.1: Utilization of the physical links.

The configuration parameters of the agents are in the policy file shown in Figure 7.3. The behaviors of the autonomic cycle run at an interval of 1s in normal state, which the use of all physical links in the network are below the threshold set in the Analyze behavior, configured at 50%. When occurs

an overload, the cycle frequency increases at a rate of 5 runs per second, i.e., an interval of 0.2s.

The graph of Figure 8.1 shows the utilization of the physical links throughout the experiment. A flow passing through the virtual network A, with constant rate of 40Mbps generated by the `iperf` application, starts close to 5s. The Fast Ethernet physical link reaches 40% of utilization, not exceeding the threshold of Analyze behavior. Another flow is started about 15s, through the virtual network B at a constant rate of 40Mbps. The link utilization exceeds the threshold, reaching 80%, and the correction begins to be performed by the Plan behavior. The ending of the execution occurs near the 18s, when the flow of the virtual network is migrated to the Giga Ethernet link. Therefore, the multi-agent system takes about 3 seconds to prepare and implement the changes.

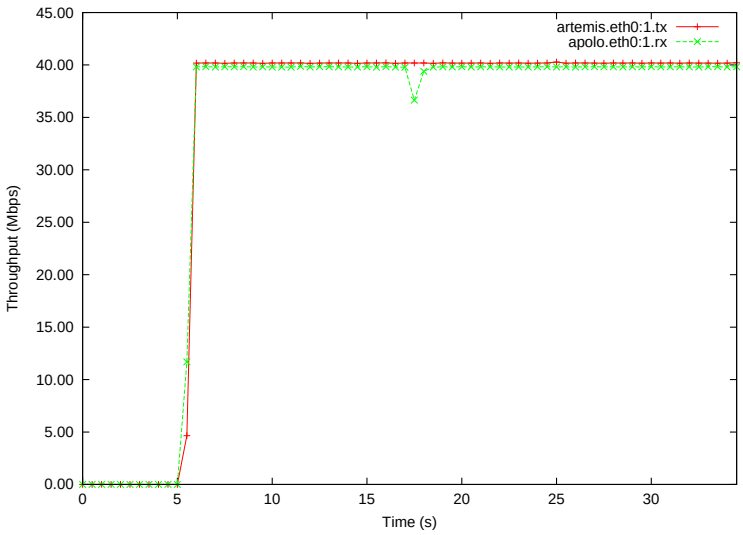

Figure 8.2: The traffic loss in the virtual network, measured on the hosts.

The graph in Figure 8.2 shows the packet loss caused by the exchange of virtual network A link mapping. The curves correspond to the sending of packets by artemis and the reception of packets by apolo. Losses can be detected when the difference between the two curves increases. In this particular experiment, the loss is registered between the time 17 and 18s. According to the `iperf` report, the amount of missing data was 1.9Mb, 158 packets of 1470 bytes, which corresponds to 47ms of loss in a traffic with a constant bitrate of 40Mbps.

Chapter 9

Conclusion and Next Steps

The Horizon project proposes a new system or plane, the Piloting Plane (PP), which enables the cooperation of the various autonomic control loops in the network, ensuring that the operates within the boundaries set by the business goals defined by the operators. This document presented the initial design of the piloting plane in the Horizon project, defining its functions, requirements and the concepts behind the operation of the components that make the piloting plane, the PAs. In addition, it provides a self-management system based on multi-agents provided by the Ginkgo Distributed Network Piloting System.

A Piloting Agent presented is a functional entity of the Horizon architecture that deals with inter-domain management tasks, such as the federation, negotiation, governance, and distribution of management domains. The APSs have two main building blocks. The first one is the Dynamic Planner, which serves as an autonomic policy-based dispatcher. The Dynamic Planner creates and destroys Behaviours, which implement the interfaces for the PAs and the core inter-management functions. The second building block represents a Behaviour, which has two functions. The first one, carried out by the Core Behaviours, is to implement piloting tasks. The second, implemented by the PA Behaviours, is to act as a proxy for the communication of the APSs with the PAs that it orchestrates.

Although not incorporating all the features designed for the Piloting Plane, the self-management system prototype implemented changes of mapping of virtual links if the load of specific physical links increases more than a certain threshold, and the results presented in Chapter 8 confirm the value of the approach.

The next steps of the work are related to the extension of the platform to implement the ideas presented in Figure 1.1: specialized agents to execute a role, like power and performance control; the existence of a hierarchy of

agents; and the management based on service level agreement.

Bibliography

[1] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, pp. 223–259, December 2006.

[2] D. Gaïti, G. Pujolle, M. Salaun, and H. Zimmermann, "Autonomous network equipments," in *Autonomic Communication*, pp. 177–185, 2006.

[3] Y. Cheng, R. Farha, M. S. Kim, A. Leon-Garcia, and J. W.-K. Hong, "A generic architecture for autonomic service and network management," *Computer Communications*, vol. 29, no. 18, no. 18, pp. 3691 – 3709, 2006.

[4] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, no. 1, pp. 41–50, 2003.

[5] Q. Mahmoud, *Cognitive Networks: Towards Self-Aware Networks*. Wiley-Interscience, 2007.

[6] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, (New York, NY, USA), pp. 3–10, ACM, 2003.

[7] J. Turner and D. Taylor, "Diversifying the internet," in *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, vol. 2, pp. 6 pp. –760, December 2005.

[8] N. Feamster, L. Gao, and J. Rexford, "How to lease the internet in your spare time," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 61–64, January 2007.

[9] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34 – 41, April 2005.

[10] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1 –12, april 2006.

[11] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 17–29, March 2008.

[12] I. Houidi, W. Louati, and D. Zeghlache, "A distributed and autonomic virtual network mapping framework," *Autonomic and Autonomous Systems, International Conference on*, vol. 0, pp. 241–247, 2008.

[13] IBM, "An architectural blueprint for autonomic computing. autonomic computing white paper fourth edition," June 2006.

[14] D. Gaiti and G. Pujolle, "Performance management issues in atm networks: traffic and congestion control," *IEEE/ACM Trans. Netw.*, vol. 4, pp. 249–257, April 1996.

[15] T. Bullot, R. Khatoun, L. Hugues, D. Gaïti, and L. Merghem-Boulahia, "A situatedness-based knowledge plane for autonomic networking," *Int. J. Netw. Manag.*, vol. 18, pp. 171–193, March 2008.

[16] G. P. T. Bullot, D. Gaïti and H. Zimmermann, "A piloting plane for controlling wireless devices," *TELECOMMUNICATION SYSTEMS*, vol. 39, pp. 195–203, October 2008.

[17] J. Strassner, N. Agoulmine, and E. Lehtihet, "Focale: A novel autonomic networking architecture," in *Proceedings of the Latin American Autonomic Computing Symposium (LAACS)*, 2006.

[18] A. Project, "Autonomic network architecture," 2011.

[19] H. Project, "A content-centric network architecture for opportunistic communication," 2011.

[20] B. Project, "Bio-inspired service evolution for the pervasive age," 2011.

[21] C. Project, "Component-ware for autonomic situation-aware communications, and dynamically adaptable services," 2011.

[22] N. Niebert, "Ambient networks: a framework for mobile network cooperation," in *Proceedings of the 1st ACM workshop on Dynamic interconnection of networks*, DIN '05, (New York, NY, USA), pp. 2–6, ACM, 2005.

[23] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "Refactoring network control and management: A case for the 4d architecture," tech. rep., 2005.

[24] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, (Berkeley, CA, USA), pp. 41–41, USENIX Association, 2005.

[25] G. Networks, "Ginkgo distributed network piloting system. white paper," 2008.

[26] I. Fajjari, M. Ayari, and G. Pujolle, "Vn-sla: A virtual network specification schema for virtual network provisioning," *International Conference on Networking*, pp. 337–342, 2010.