Une approche pour la composition autonome de services de communication orientés QoS. Application aux protocoles de transport configurables

Nicolas van Wambeke

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Institut National des Sciences Appliquées*
Discipline: *Informatique et Télécommunications*

Présentée et soutenue par NICOLAS VAN WAMBEKE
Le *08 Septembre 2009*

Une approche pour la composition autonome de services de communication orientés QoS
Application aux protocoles de transport configurables

JURY

M. Paul AMER	Rapporteur
Mme. Pascale VICAT-BLANC PRIMET	Rapporteur
M. Serge FDIDA	Examinateur
M. Flavio OQUENDO	Examinateur
M. Michel DIAZ	Invité
M. Khalil DRIRA	Invité
M. Christophe CHASSOT	Directeur de Thèse
M. Ernesto EXPOSITO	Directeur de Thèse

Ecole doctorale: *Mathématiques Informatique Télécommunications de Toulouse (MITT)*
Unité de recherche: *Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS)*
Directeurs de Thèse: *M. Christophe CHASSOT et M. Ernesto EXPOSITO*
Rapporteurs: *M. Paul AMER, Mme. Pascale VICAT-BLANC PRIMET*

A Mathias

REMERCIEMENTS

Les travaux présentés dans ce manuscrit ont été effectués au sein du Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS). Durant mon séjour, le laboratoire a été dirigé successivement par Monsieur M. Ghallab et Monsieur R. Chatila que je tiens à remercier pour leur accueil.

Je souhaite également remercier Monsieur F. Vernadat, responsable du groupe Outils et Logiciels pour la Communication (OLC) pour son accueil au sein de l'équipe.

Mes travaux ont été encadrés par Messieurs E. Expósito et C. Chassot a qui je souhaite témoigner ma plus sincère gratitude pour leur écoute et leurs conseils tout au long de ces trois années. Je les remercie tout particulièrement pour les nombreuses réflexions que nous avons pu mener ensembles et au travers desquelles ils ont partagé une partie de leur expérience avec moi.

Je tiens également à remercier Messieurs P. Amer, M. Diaz, K. Drira, S. Fdida et F. Oquendo pour leur participation à mon jury de thèse. Aussi, je souhaite remercier tout particulièrement Madame P. Vicat-Blanc Primet et Monsieur P. Amer de m'avoir fait l'honneur d'être rapporteur de mes travaux. Leurs remarques et conseils avisés ayant permis d'améliorer la clarté de la présentation des idées véhiculées par le présent manuscrit.

Au cours de ces trois années, j'ai été amené à travailler dans les projets européens EuQoS et NetQoS. Je souhaite remercier les acteurs qui ont rendu possible ces collaborations fort enrichissantes. Plus particulièrement, je souhaite remercier Christof Brandauer (Salzburg Research) pour la richesse des échanges que nous avons eus sur le plan technique et dont l'outil MINER a grandement facilité l'évaluation de mes propositions.

Je souhaite remercier mes collègues et amis pour leur bonne humeur et leur soutien dans les moments de doute et d'indécision (dans l'ordre alphabétique): Ion, Vincent, François, Mehdi, Claire, Fred, Mathieu, Julien, Jorge, Vincent, Fabien, Clémence, Renaud, Florin, Pauline, Enguerran, Ana, Siegfried, Louise.

Merci à tous ceux qui ont su m'accompagner et m'épauler pendant ces trois années, je pense ici en particulier à ma famille et à mes amis qui ont su apporter leur soutien inconditionnel.

Enfin, je ne saurai jamais assez remercier ma femme, Annetta, pour son encouragement, son soutien mais également sa patience, sa douceur et sa compréhension.

List of Acronyms

- **ACT** : Autonomic Communication Toolkit
- **ASM** : Autonomic Service Manager
- **ATP** : Autonomic Transport Protocol
- **AUP** : Agile Unified Process
- **PDM** : Platform Dependant Model
- **PIM** : Platform Independent Model
- **MDP** : Markov Decision Process
- **MM** : Machine Manager
- **QoS** : Quality of Service
- **RL** : Reinforcement Learning
- **TACT** : Transport layer Autonomic Communication Toolkit
- **UML** : Unified Modeling Language

CONTENTS

GENERAL INTRODUCTION

> *Quality in a service or product is not what you put into it.*
> *It is what the client or customer gets out of it.*
>
> Peter Drucker

RECENT years have witnessed many changes in the way the Internet is brought to end users. All around the globe, from slow 56Kbps telephone lines to high speed hundreds of megabits/s fiber access, users have seen their connection speed increase drastically. In parallel, the applications that are used on end hosts have exploited the computational power made available to them by handling an increasing number of complex medias and tasks. From binary and text transfer applications to live audio and video streaming, the requirements of content distribution have changed in many ways.

The early communication model based on the Transmission Control and Internet Protocols (TCP/IP) was well suited in the early days of the Internet. The limits of this architecture for today's applications have been demonstrated by the scientific community, thus giving birth to the field of Quality of Service (QoS) in networks.

Context and problem statement

The increasing number of multimedia applications that are deployed over the Internet has motivated the need for solutions to performance issues that affect a user's experience. These issues can be classified in two main categories: quality of service and coordination.

Quality of Service

The performance problems that affect novel applications arise from the fact that they required the network over which they were executed to provide them some temporal as well as reliability guarantees

on the service. These new requirements being sometimes of dynamic nature as they vary throughout the life of the communication.

The most well-known solutions are certainly at the IP layer where these approaches led to the Integrated and the Differentiated Services models in order to provide applicative flows transiting through a network (an Internet autonomous system) with different priorities on their treatment. Furthermore, several projects have taken the approach one step further with the advent of multi-domains end-to-end QoS management frameworks and protocols for the management of such systems.

However, the field of Quality of Service research is not limited to the IP domain. For example, the 802.11e standard which defines a set of mechanisms to be implemented at the MAC layer by wireless devices to provide differentiated access categories depending on the nature and requirements of the flow is another example of solutions that aim to provide QoS. Similarly, other solutions at the MAC layer are targeted towards supporting QoS in different environments such as wired (802.1p standard) or ad-hoc networks. At the transport level, multiple approaches exist to optimize the QoS provided to applications. Several of these approaches will be presented in the following chapters.

Coordination

The existence of multiple solutions for providing Quality of Service at multiple layers of the communication stack mentionned above raises the question of which solution to choose. Furthermore, these solutions are generally specific to both the access network technology and the type of media that is being transported. These conditions raise the need for coordination to ensure the coherence of choices. Indeed, applying QoS mechanisms at various layers without coordination can lead to a number of complex scenarios in which the effects are counter productive resulting in a less satisfactory overall user experience.

The coordination issues have led to multiple proposals being formulated. These solutions mostly rely on cross-layering techniques which allow a protocol being executed at a given layer (n) to take information produced by some other layer in the stack into account. It is foreseeable that multiple solutions for providing QoS at one layer could co-exist on a single end-host. The decision of which solution to be implemented would then be taken based on cross-layer knowledge as well as in

concertation with a coordinator entity. The autonomic computing framework provides architectural guidelines for systems that present such characteristics as it allows for a hierarchical organization of the decision process.

Providing Quality of Service support in networks in a coordinated fashion is a key element to a sustainable growth of the service that the Internet can support. However, in order for these solutions to be deployed, the features that enable "global" coordinated adaptation have to be defined.

Contributions of this work

This thesis presents the design and implementation of the *Autonomic Communication Toolkit (ACT)*, a connection management system for end hosts at multiple levels following the autonomic computing paradigm to provide applications with communication channels which best match their QoS requirements. Management is performed at multiple levels which are hierarchically ordered. Respectively the Machine, Application and Communication Channel levels are taken into account. User preferences are captured at the Machine level and are combined with application developer's preferences specified at the Application level to provide guidelines to Communication Channel managers. These channels are responsible for handling the data produced by the application.

Based on the previously presented limitations of current communication systems, the work presented herein aims at providing effective models, guidelines and a prototype implementation that can be easily used to enhance existing systems. For this, several requirements for the contribution have been isolated:

1. The solution should be deployed on the end hosts exclusively. It should not require the presence of "in network" devices to be deployed by the operator or service provider.

2. No assumption should be made on the network environment in which the system evolves.

3. Any existing or new application should be able to use the system.

4. Any available *service* on the host should be usable for communications managed by the system.

5. Adding new functionalities to the system should be possible without compromising the system's efficiency.

Structure of the dissertation

The dissertation is structured in four chapters. After presenting the state of the art and context in which the work takes place, the Autonomic Communication Toolkit (ACT) is presented as a general model for a multi-layer, user-centric coordinated communication system design. An implementation of the ACT model at the transport layer (TACT) is presented and evaluated as a proof of concept.

Chapter 1 presents the state of the art and context in which the present contributions take place. After studying the various existing approaches to provide Quality of Service for end-hosts, a modular approach to communication protocol design is presented as an effective approach to adaptation at the transport layer. To support coordinated adaptation of modular components, the Autonomic Computing Model is introduced as an architectural reference.

Chapter 2 details the main contribution of this work. Indeed, the design of the Autonomic Communication Toolkit is presented. The presentation follows a well known requirements capture and system design method that leads to a "Platform Independent Model" (PIM) of the proposed solution. The modeled system allows to manage adaptation of communication protocols at multiple layers following an autonomic approach. A key question in adaptation is "how to adapt?". Artificial Intelligence techniques such as Markov decision processes and reinforcement learning are explored to provide possible answers to this question.

Chapter 3 describes the implementation of the ACT at the transport layer (TACT). The approach consists in refining the previously presented model to discuss its support for modular transport protocols. The limitations of the ACT are identified and solutions for overcoming them are presented.

Chapter 4 presents an evaluation of the approach at both functional and non-functional levels. A first simulation study provides a proof of concept of the approach. Further experiments involving a

Java prototype implementation of the system are presented in order to illustrate the learning and decision elements in action. A discussion on the scalability and extensibility of the approach is presented.

Finally, conclusions to this work are presented and several research perspectives extending specific aspects of the present work are identified.

Reading Guide

When used in the margin of a text box like this one, this symbol denotes a bibliographical precision.

When used in the margin of a text box like this one, this symbol points out that the contents are of a technical nature but not mandatory to obtain a global understanding of the contribution itself.

CHAPTER 1

CONTEXT AND STATE OF THE ART

I have but one lamp by which my feet are guided, and that is the lamp of experience. I know no way of judging of the future but by the past.

Edward Gibbon

Introduction

In this chapter, the elements which inspire the contributions of this thesis are presented. A broad range of topics are visited to explore several fields of interest and present great research work that are now part of the State of the Art in these topics.

A first section is centered on the technologies available to provide either guaranteed or optimized Quality of Service (QoS) to end-hosts. The focus is made on the parts of these systems that are visible to applications or entities executing on the end-hosts. Solutions are considered both at the transport and Network Layer as these are the layers which span across the network to provide end-to-end services.

The numerous different implementations of Transport Layer mechanisms have led to the emergence of Micro-Protocol composition frameworks. A second section will take us into the study of these frameworks which all follow different approaches to fine grained, on demand, protocol service establishment by assembling simple building blocks called Micro-Protocols. These frameworks provide the tools for building communication protocols that work in diverse environments providing that

they have all the required Micro-Protocols and "know" how to assemble them together.

Facing the high number of services available for either optimizing or guaranteeing QoS at multiple layers and even inside the layers themselves, a third section presents existing approaches to coordinating the instantiation of these services. The strengths and limitations of these approaches are presented.

A fourth section presents architectures that provide possible answers to the problem of service composition. The approach is taken from the field of software engineering. Autonomic Computing provides architectural guidelines to build multi-tiers systems composed of components that take their own decisions based on the data they collect or that they acquire. Autonomic approaches are interesting as they perform better than centralized approaches in terms of scalability. This improvement is due to the fact that the decision on how to fix a given problem is always taken as close as possible to its source in the architecture.

Finally, definitions of the concepts that will be used in the description of the contribution in the following chapters are given. A partial conclusion summarizes the key concepts of this chapter and introduces the structure of the following chapters.

1.1 QoS for the end-hosts

Providing Quality of service (QoS) for data flows on the Internet has been a major goal of the past decade. Indeed, the increasing demand for transit and relay bandwidth in core networks combined with the massive distribution of enterprise and multimedia applications have led to reconsidering the services offered by major service providers.

Indeed, one of the characteristics of the Internet and networking in general relies on the fact that most of the communications occur following the "Best Effort" paradigm. In this service model, the operator provides no guarantees on what the user gets out of its access. Indeed, for Internet connectivity, a user may lease a T1 operating at "1.544Mbps" or even an OC-3 operating at "155.52Mbps" and not be able to achieve throughputs greater than a few Kbps for some destinations on the Internet. The customer is only sure of the maximum bandwidth to and from any destination which is the speed

of its leased link.

The causes of these phenomena are outside of the scope of the current study but are of much interest. Mainly, the heterogeneous and statistical nature of the Internet, the administrative preferences of "manually tuned" routing protocols (BGP) and poor engineering are responsible for these situations.

Given the above observation, most network operators use traffic engineering techniques combined with some level of over-provisioning to provide acceptable services to their users. The scientific community has followed two main tracks in its research to improve the situation.

- A first and straightforward approach is based on the assumption that finding a global solution to the problem in the actual Internet is not feasible. This approach consists of trying to optimize the use of the resources available from the network at a given time. In this context, no delay or bandwidth guarantees can be obtained but still, improvements can be achieved. For example, in the context of a video transmission over the Internet, a simple optimization strategy consists in sending only the audio stream when there is not enough bandwidth for both audio and video. This decision degrades the experience of the user but not as much as it would be if an unintelligible fraction of both video and audio were received.

- A second family of approaches consists in looking for ways to provide guarantees to the service that users get from the network. For example, instead of leasing a T1, the user and the operator engage in a contract accompanying the connection in which the service is best described. This is known as a Service Level Agreement or SLA. In the SLA, the operator describes the service that the user will get out of the network as well as reimbursements that the user will receive in case the service level is not met. For example, the operator might lease a T1 to a customer with an SLA that can be summarized as follows: *on demand max 6 concurrent reservations, per reservation, at most 128Kbps as guaranteed minimum, always 512Kbps burstable, delay and losses at 99th percentile below 50ms and* 10^{-7} *to the following destinations: 200.120.192.0/22 and 7.0.0.2/8, "Best Effort" for all other traffic or traffic for which no reservation is made*.

This section studies the technologies for achieving both kinds of service. For each of these methods and technologies, the focus is made on the end-host observable part of the system with little detail

on the methods used to actually provide guarantees in the network.

1.1.1 Techniques for optimized QoS

Optimizing the QoS for the end-users is an interesting technique which finds its use in many cases as soon as the bandwidth available on the data path is less than the bandwidth required to send all data in a timely manner. These situations are frequent in the best effort Internet but can also occur in the guaranteed world. Indeed, many factors interplay such as, for example, the unavailability of resources at the moment they are needed, or the technical impossibility to provide guaranteed services due to the lack of adequate technology being deployed.

The following section details the optimization techniques that exist at the various layers of the communication stack from the application itself to the Transport and Network Layers.

1.1.1.1 By the multimedia applications

Applications which use the network to transfer multimedia contents have become increasively interested in optimization techniques. This is mainly due to the importance of having the data they produce delivered to their peer using services that did “more than UDP” but “less than TCP”. With the advent of Micro-Protocol composition frameworks (presented in detail in section 1.2.2.3) however, most of the techniques originally implemented by the application themselves can now be handled by the transport protocols themselves.

Indeed, TCP provides too many guarantees to these applications. Or more precisely, the full reliability that is provided by TCP leads to an inability to bound the end-to-end delay which makes the service unusable by real time multimedia applications such as video-conferencing. On the other hand, UDP is also insufficient because it provides a service which does not take advantage of the fact that some applications can tolerate a few retransmissions in case a loss occurs as long as the delay is kept below a certain threshold. These limitations led to the implementations of custom ARQ (Automated Repeat reQuest) mechanisms on top of the services provided by UDP by some applications.

Another common approach in the case of multimedia applications consists in monitoring the services provided by the network at all times and using “adaptive coding” [M-17] which adaptively

changes the coding scheme (the codec) used to encode the media that are being transmitted. Generally speaking, the use of a codec that requires less bandwidth will lead to a quality decrease for the user but will achieve better results than dropping half of the packets. Modern codecs and frameworks for multimedia coding and encapsulations such as the MPEG-4 norm define several levels of coding. Additionally, to avoid the signalling overhead that a change of codec requires, hierarchical codecs have been developed. These codecs allow for the quality of the recomposed video to be proportional to the number of received packets given that the discarded packets are chosen carefully.

1.1.1.2 At the Transport Layer

Quality of service can also be optimized at the Transport Layer. This is generally performed by the enhancement of one of the Transport Layer functionalities. In [M-16] the following classification of the basic Transport Layer mechanisms is presented.

- **Error control:** combination of techniques to protect against loss or damage of user data and control information. Error control is performed in two phases: error detection and error reporting and recovery. Error detection identifies lost, disordered, duplicated and corrupted Transport Protocol Data Units (TPDUs). Error reporting is a mechanism where the transport receiver informs the sender about errors detected. Error reporting may be implemented by positive acknowledgment of data received (ACK) or negative acknowledgment of errors detected (NACK). Error recovery is a mechanism used by the sender or receiver to recover from errors. Error recovery mechanisms can be implemented by retransmission strategies (i.e., Automatic Repeat reQuest or ARQ) or redundancy mechanisms (i.e., Forward Error Correction or FEC).

- **Flow and congestion control:** flow control is a mechanism implemented by the Transport Layer to limit the rate at which data is sent over the network to avoid exceeding the capacity of the Transport Layer receiving entity. Congestion control mechanisms are intended to preserve network resources to avoid network congestion.

- **Multiplexing and demultiplexing:** mechanisms implemented by a transport protocol to associate several Transport Service Access Points (TSAPs) to a single Network service Access

Point (NSAP). These mechanisms enable supporting several Transport Layer connections using the same Network Layer connection. The use of protocol port numbers to perform the multiplexing/demultiplexing operations allows serving multiple transport users using the same network address.

- **Segmentation and reassembly:** when the size of the Transport Service Data Units (TSDUs) is bigger than the allowed size of the Network Service Data Units (NSDUs), the TSDUs have to be segmented into smaller TPDUs by the transport sender. The transport receiver reassembles the TPDUs to rebuild the TSDUs to be delivered to the user receiver.

- **Other mechanisms:** some transport protocols can implement other specialized mechanism such as concatenation/separation. The concatenation combines several TPDUs into one NSDU to reduce the number of NSDUs submitted to the Network Layer; the separation of the concatenated TPDUs will be performed by the transport receiver entity.

To improve the Quality of service for multimedia applications, the error control and congestion control mechanisms can be extended. The following paragraphs show some of these extensions which are implemented in the ETP framework which is presented in detail in section 1.2.3.

Specifically, the following paragraphs present an enhancement of the congestion and error control mechanisms to make them more compliant with the QoS requirements of multimedia applications while taking into account limited network services.

1.1.1.2.1 QoS-aware congestion control

Currently, most streaming and conferencing applications do not implement congestion control mechanisms. The implementation of a congestion control mechanism for these applications is required to reduce the risk of future Internet congestion collapse [M-9]. Real-time flows are either transmitted using a near-constant rate or an adjustable rate based on the feedback obtained from the receiving application (i.e., RTCP messages). But even when applications are able to adapt their sending rate, it is usually done in long timescales. Some studies have led to the design of conges-

tion control mechanisms adapted to the characteristics of these applications [M-10]. One of these mechanisms is the TCP-Friendly Rate Control or TFRC and its extensions for the wireless context [M-6, M-22]

The rate control mechanism of TFRC is based on a delaying policy aimed at adapting the flow to the allowed sending rate. This mechanism can penalize applications with strict delay constraints as some received packets could be discarded as they arrive too late to be presented. Approaches such as the ones proposed in [AAI-9] try to optimize the quality of the video stream by selectively discarding packets at the Transport Layer. To achieve this, QoS information describing the multimedia flows has to be made available to the Transport Layer. This can be achieved by using mechanisms similar to the ones described in [AAI-8].

These selective ADU (Application Data Unit) discarding methods can be applied if the media has been encoded using specific codecs which facilitate the implementation of this method at the Transport Layer (i.e., ALF approach for the segmentation of flows such as MPEG [F-7], H.263, MJPEG, etc.). Similarly to the multi-layered multicast flows, for Transport level quality adaptation strategy, several "quality layers" could be defined using the QoS description of the ADUs composing the multimedia flows. For instance, for an MPEG flow composed by I, P and B images, three quality layers could be defined: *Layer 2:* I, P and B images - *Layer 1:* only I and P images - *Layer 0:* only I images.

1.1.1.2.2 QoS-aware error control

Some multimedia applications have a preference for timeliness over order and reliability (i.e., video conferencing). Actually, many of these applications do not require a fully ordered and fully reliable transport service but can acomodate partial order delivery [F-2]. For this reason, most of multimedia applications have been designed to use the UDP protocol without any guarantees of order and reliability.

ARQ (Automated Repeat reQuest) error control mechanisms work as follows: when a loss is detected, the receiver sends a feedback message to ask the source to retransmit the message. This

means that a retransmitted packet arrives at least three one-way delay after the transmission of the original packet. Sometimes, this delay could exceed the delay constraints of the application. However, if the one-way delay is short, this mechanism could be efficiently used to recover the losses.

When the mechanism is receiver-based, its objective is to avoid retransmission requests when data will not arrive on time for its presentation. In order for this method to be implemented, the receiver has to know the scheduled presentation time of the lost packets.

On the other hand, the request can be sender-based, to avoid retransmissions of packets that will arrive too late to be presented. The retransmissions can be demanded by the receiver when losses are detected.

This mechanism can be easily implemented by the source if QoS information related to the time presentation is available at the Transport Layer. Indeed this method can be used to provide a differentiated and partially reliable service taking into account the notion of differentiated layers previously introduced.

1.1.1.3 At the Network Layer

Optimizations of the QoS at the application and the Transport Layer can only be performed on end-hosts or eventually by transport proxies in the network. For example, in the case of a mechanism that takes the available bandwidth on the path into account to discard low priority ADUs, the end system must estimate the bandwidth that is available at any given time to determine which packets it will drop and which it will keep. If the selective discarding mechanism was implemented on each of the nodes through the network, the bandwidth estimation would not be required as the decision to drop low priority packets in favor of high priority ones could be made on every node when a buffer overflows. This approach is discussed in [M-11].

This behavior can be achieved by the use of scheduling techniques in the management of the buffers at the IP layer. For instance, early studies that dealt with QoS at the IP layer have introduced the need for efficient scheduling algorithms which would allow for priorities to be taken into account.

The priority of an IP packet is determined by using the value of its ToS (Type of Service) header field (now extended and renamed DSCP for Differentiated Service Code Point). The IETF has defined

standard values for the DSCP to be used on the Internet [M-19]. Depending on these values, the routers will handle packets differently to provide a specific service.

Among the different values of the DSCP field, a packet could be tagged with Minimize Delay (0x10) which would lead to a treatment among its route for it to arrive as fast as possible to its destination. However, the packets tagged this way may experience more losses than the ones marked as Maximize Reliability (0x04) which would lead to less losses but might increase the delay.

The value of the DSCP field can be set by the sender of the packet. However, in the current Internet, there are no guarantees that the various organizations that control the routers which will handle the packet will actually use these values to provide differentiated yet unguaranteed services. Additionally, any of the equipments that handle the data in the network may reset the value or change it without the sender knowing which might lead to unexpected results.

1.1.2 Techniques for Guaranteed QoS

When optimizing the QoS is not enough, or in environments where there is more control on the intermediate equipments and systems that manage packets enroute, QoS guarantees can be obtained. The solution for providing these guarantees consists in combining a specific packet scheduling technique with an admission control. For example, we've seen in the previous section (1.1.1.3) that scheduling algorithms provide for a way to allocate a portion of the outgoing link bandwidth on a networking device to a given type of packets. If these packets are marked when they enter the network and that prior to their marking, a procedure has been executed to verify that the service required for the connection to take place can be provided and assured, the QoS is guaranteed.

The following sections present two of the various systems that implement guaranteed QoS with or without explicit prior reservation in either mono or multi domain environments. These systems rely on frameworks such as the IntServ [M-3] and the later DiffServ [M-2] models promoted by the IETF for achieving their goals.

1.1.2.1 The EuQoS system

The EuQoS System focuses on providing end-to-end guaranteed QoS in heterogeneous multi-domain contexts. The EuQoS approach is detailed in [M-5]. In EuQoS, a virtual layer is added to make the management of resources independent of the technology that will be used to provide them. Additionally, the multi-domain context which is targeted in EuQoS requires the use of a signalling protocol as identified in [M-8] to establish guarantees on all the networks on the data path.

From an end-user and an application programmer's point of view, the EuQoS system provides tools that allow reservations to be made. While the user is presented with a nice interface, the programmer will be happy to see that an API (Application Programming Interface) is also provided. When a communication that requires guarantees is about to begin, EuQoS must be invoked by providing it with the following parameters: Source IP Address, Destination IP Address, Source Port Number, Destination Port Number, Transport Protocol, Required Peak Bandwidth and Maximum delay or Applicative Codec.

After processing, the system will return the description of the service it has been able to reserve along the path. At best, this service will match the requirements passed at invocation but can also be different in case there is not enough resources on the data path or the requested resources exceed prior agreements between the user and the provider.

1.1.2.2 The NetQoS system

The NetQoS system is designed to ease the management of a QoS-enabled domain such as the ones using the EuQoS framework. NetQoS performs automated reservation and configuration of resources based on a policy based mechanism. The system is based on the initial view presented in [M-20].

In NetQoS, users and applications can specify their requirements to the system. These requirements constitute a high level policy (i.e., "When I'm at work, I like having the highest possible quality for my VoIP phone calls", "on the other hand, at home, I don't mind getting bad quality if it allows me to pay less").

Based on the user defined high level policies, the system will derive an intermediate policy which corresponds to the translation and mapping of the high level policies into Quality of service param-

eters such as delay, loss rate and throughput. This intermediate policy will then be used to perform admission control and eventually change the configuration of the policy enforcement points (PEP) at both the network and Transport Layers.

From an application developer's point of view, the NetQoS system offers an API that allows it to specify the actual QoS requirements for its streams. The system will provide the user with a service that best suits his needs and its contract. Additionally, the system will allow the user or application to access statistics information about the service it is getting.

Additionally, the NetQoS system provides support for the policy based management of QoS services at multiple layers of the communication stack. The policies that can be expressed by the different actors in the framework are not only translated to operational directives for the network layer. The use of Micro-Protocol composition frameworks are the Transport Layer is also supported. The study of these frameworks illustrating the possibilities that they provide in terms of QoS is detailed in the next section.

1.2 Micro-Protocol composition frameworks

1.2.1 Motivations

Early distributed applications had simple (i.e., total or null) requirements regarding the reliability and order of the data they exchanged. The widely used transport services such as UDP and TCP have been designed to fulfill such simple requirements.

Today, new applications exchanging different kinds of data such as audio or video presentations are becoming increasingly popular over the Internet (VoIP, VoD, TVoD, ...). These applications have different requirements than the "historical" ones. In order to provide adequate services, several proposals have been made at the network level (DiffServ, IntServ, ...). However, when these cannot provide any guarantees (i.e., in wireless environments), the Transport Layer is able to implement mechanisms that "optimize" the QoS as described in [M-12]. In parallel to several proposals made at the IETF such as DCCP [F-10] and SCTP [F-16] to tackle specific aspects of the problem, modular protocols such as Coyote [F-4] and ADAPTIVE [F-15] have been proposed.

This chapter also presents the ETP (Enhanced Transport Protocol) which is specifically designed to provide a large set of transport mechanisms to most efficiently satisfy the requirements of a wide range of applications.

1.2.2 Background

Given the limits that the tight coupling of existing transport protocol's functions, the extensibility of such protocols to take into account new application requirements or underlying network services is a complex task [F-1]. Indeed, to provide a service that best suits the application requirements, the internal architecture could be modified (e.g. replace an ARQ error control mechanism by a FEC mechanism) [F-5, F-14]. Different compositional frameworks have been proposed to deploy such communication architectures.

The most intuitive model for the internal architecture of such protocols is the OSI model. Indeed, it is easy to consider the Transport Layer as being composed by a hierarchical stack of *blocks*. Each of these blocks providing a specific elementary service. This hierarchical architecture is followed by V_STREAMS [F-13], the X-kernel [F-9] and APPIA [F-11, F-12, AAI-21] detailed hereafter.

Another approach consists in deploying an event based system in which all the different *blocks* are triggered by a global *kernel* in response to specific events that take place throughout the message processing. For example, several modules can subscribe to the event *arrival of a new applicative message*. The kernel will then invoke them each time this event is triggered. This is the approach followed by Coyote [F-4],Cactus [F-17, F-18, F-5] and ADAPTIVE [F-14, F-15] presented hereafter.

1.2.2.1 The Micro-Protocol abstraction

Most of the proposed architectural solutions for dynamically composable protocols are based on the Micro-Protocol abstraction. This notion was first used in x-kernel. Its essence comes from the decomposition of the different functions of a protocol into simple elementary functions. The protocol being implemented by the successive call of *primitive building blocks* [F-9]. Following the x-kernel, the Micro-Protocol notion is also used by ADAPTIVE [F-15] in which a protocol is implemented as the composition of multiple reusable *building blocks*.

This notion has been further enhanced by the Coyote and Cactus projects. A Micro-Protocol is a software module implementing a standard interface allowing it to interact with other Micro-Protocols. This module implements an elementary function required for the deployment of a communication protocol.

Micro-Protocols are generally part of a *framework* which includes the *kernel* and several basic helper objects. In these frameworks, the Micro-Protocols act as off-the-shelf components coordinated by the *kernel*.

1.2.2.2 Event based frameworks

Event based architectures have been investigated in the context of several research projects that have led to the development of new transport protocol's architectures. In these systems, the *send()* system call of the API triggers a specifically predefined event to which one or more Micro-Protocols have subscribed. These Micro-Protocols can also trigger new events which will be handled by others as shown on Figure 1.1. In what follows, various approaches to event based architectures are presented.

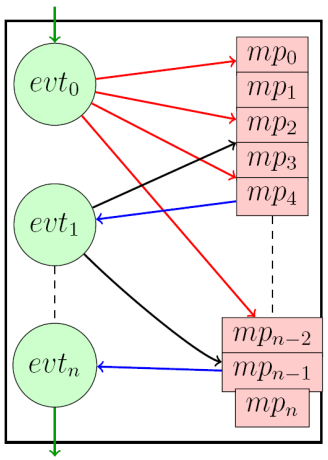


Figure 1.1: Event based architectures

Coyote [F-4] extends the model provided by x-kernel by increasing its granularity. By such, intra-layer composition becomes possible and the system is composed by a central controller which coordinates the message handling. Various type of elements are defined: *system events* and *user events*.

System events: these events are predefined and automatically triggered by the system's kernel when specific actions occur. For example, new data arriving from the lower layers.

User events: these events are defined by the framework user (i.e., the Micro-Protocol architects). They can only be triggered by Micro-Protocols. These events provide an easy way for a Micro-Protocol to signal the expiration of a timer to others.

A Micro-Protocol is then defined by a set of functions, each of them being the handler for a particular event. Upon instantiation, a Micro-Protocol requests the kernel to bind specific events to these handlers. The concatenation of the actions that each Micro-Protocol will take in response to these events will allow the protocol to provide specifically targeted services. The coherence of the provided service will depend on the composition choices and the architect's abilities.

The main interest of the Coyote framework resides in its modularity that allows heavy reuse of modules to provide numerous services to the applications.

Cactus [F-5] extends the model proposed in Coyote to include new mechanisms specially dedicated to QoS for applications. Transport protocols are by nature, unable to provide any guarantee on QoS. The only possibility resides in the modification of the implemented services to optimize the QoS provided to applications. This is the approach that is followed in Cactus.

The Coyote system has been extended to integrate a slightly complexified structure in which the links between Micro-Protocols and events can be modified during the execution [F-17]. This allows a mechanism that is judged inefficient regarding the goals of an application to be replaced by another in a seamless way.

The limits of Cactus are related to its structure which does not allow for dynamically extensible protocols to be defined [F-18]. Indeed, Cactus allows for defining bundles of Micro-Protocols which can be used at run-time to provide service adaptation. However, the specialization of a protocol for it to take new contexts into account (all possible adaptation actions being impossible to think of during its design) is not provided without having the architect re-design its protocol and re-compile it.

1.2.2.3 Hierarchical frameworks

In parallel of event based architectural frameworks that may seem complex regarding the history of protocol design, another approach to Micro-Protocol architectures exists. These architectures can be seen as a generalization of the OSI model as Micro-Protocols are stacked together to form a given protocol as depicted on figure 1.2. In what follows, these approaches are presented in further details.

The V_STREAMS was one of the first systems supporting a hierarchical and compositional model [F-13]. A stream is a full duplex connection between several linearly connected processing modules

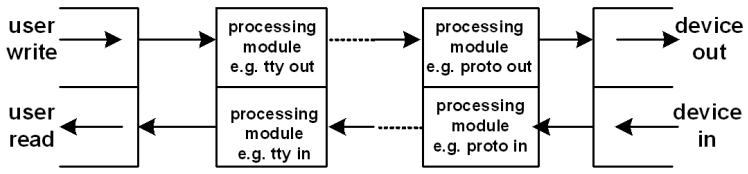


Figure 1.3: Example of a V_STREAM hierarchical architecture

with data flowing in both directions. The modules in a stream communicate almost exclusively by passing messages to their neighbors, except for some conventional variables used for some control functions. The API services consist in write and read messages. Each processing module consists of a pair of queues, one for each direction. In the V_STREAMS system a stream may be dynamically extended by the addition of new modules. In fact, the processing modules in a stream are thought as a stack whose top is next to the user program. In next figure shows a V_STREAMS configuration for network terminals.

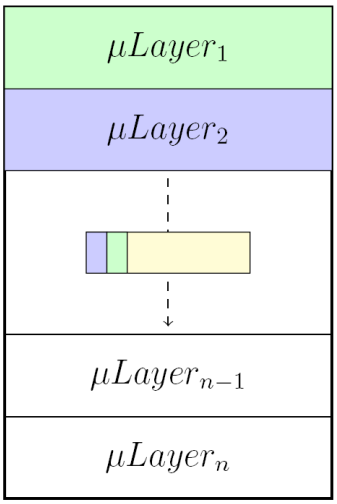


Figure 1.2: Hierarchical Architectures

The X-kernel [F-9] is a network oriented operating system. Its architecture allows the user to implement and deploy its own communication protocols (see figure 1.4. Following this approach, a user can deploy various communication protocols that, when composed together, provide a specific service to the applications using them.

Although the x-kernel was not intended to be used as a composition framework it was the base to various preliminary experiments for both the Coyote and Cactus projects. The composition logic allowed for the easy decomposing of complex protocols into *Micro-Protocols*.

ADAPTIVE [F-14, F-15] follows the same goals as the ones expressed by Cactus. However, its interest resides in the fact that it intends to develop many tools aimed not only to provide an easy deployment of custom protocols but also to evaluate the performances of such protocols [F-3]. Moreover, the different

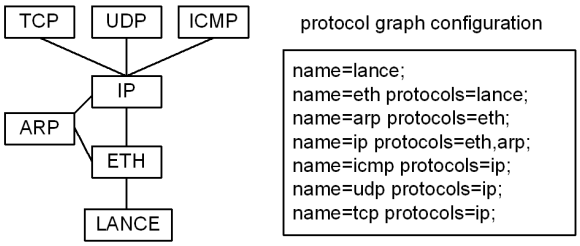


Figure 1.4: Example of an x-kernel protocol graph configuration

features (Micro-Protocols) of the constructed protocols can be activated or deactivated upon instantiation.

In ADAPTIVE, Micro-Protocols are materialized by different implementations of predefined functions related to connection or context management, reliability and transmission control, ... Upon instantiation, the application specifies its QoS requirements. The ADAPTIVE framework then chooses the best possible composition of Micro-Protocols for each predefined function by taking into account the network context described by the protocol architects.

However, ADAPTIVE does not provide mechanisms to perform dynamic reconfiguration of the instances during the protocol's execution. Moreover, the addition of Micro-Protocols or new functions cannot be performed in run-time which makes the model quite static.

An extension to ADAPTIVE known as *ADAPTIVE Communication Environment (ACE)* provides tools for automated build of network daemons.

APPIA [F-12] is based on the dynamic composition architecture as existing in the Ensemble project [F-6]. APPIA extends the model to make it more flexible. The composition of Micro-Protocols is made in a hierarchical way while still allowing for various Micro-Protocols to be instantiated at a given level [F-11]. By such, a message can be *routed* adequately through the sequence of Micro-Protocols corresponding the the composition which will allow to provide the best QoS. In APPIA, this Micro-Protocol sequence is known as a *channel*.

In addition to this, the messages exchanges in the Micro-Protocol's stack can be of two kind:

Internal Message: these messages are required to perform coordination actions among the different Micro-Protocols present in the stack. They are exchanged only between two Micro-Protocol

instances in the stack and never leave it. New Micro-Protocols can define their own messages by inheritance while still allowing older modules to interpret parts of the messages given their polymorphic nature.

Messages to be transmitted: these messages extend internal messages. By such, they still have the same properties as the internal messages but may also be transmitted outside of the Micro-Protocol stack, either to other communication protocols on the machine or to the network.

1.2.3 The Enhanced Transport Protocol (ETP)

The Enhanced Transport Protocol (ETP) [F-8] is a QoS-oriented modular transport framework following an approach combining the benefits of both the event based and the hierarchical architectures. Its internal architecture is presented on Figure 1.5

On the one hand, the hierarchical approach is good for mechanisms that require to be executed synchroneously to the stream they transport. This key concept can be applied to QoS control functions to be performed synchronously to the flow exchange between applications (e.g. source and receiver based mechanisms to flow and congestion control, error detection, ...).

On the other hand, some Micro-Protocol functions can be performed asynchronously to the stream, when a specific event is triggered (e.g. error recovery mechanism triggered by a timeout). These management functions are better implemented in an event based architecture. However, data often needs to be exchanged between the data processing and connection management functions.This is achieved by the use of a shared variables container. The object-oriented techniques that exist today provide for easy extension and evolution of the architecture.

In summary, the ETP architecture follows a hierarchical model for the composition of services related to data processing mechanisms and an event based model for the management mechanisms (see figure 1.5).

In ETP, a Micro-Protocol (e.g. implementing the TFRC congestion control [M-13]) is further decomposed into several processing modules, each being instantiated in the appropriate containers (In flow, Out flow, Management). The ETP kernel provides methods for processing modules synchronization as well as communication through the use of shared variables.

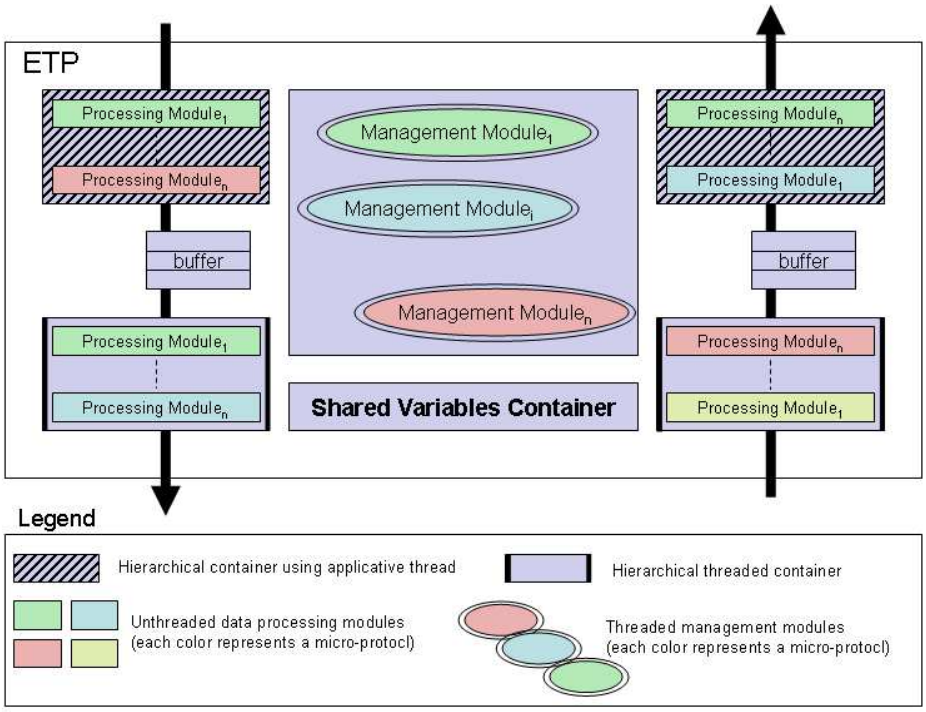


Figure 1.5: Composition architecture of the ETP

The application using ETP can specify the Micro-Protocol composition it requires and the ETP kernel will use the Micro-Protocol descriptions to instantiate the correct processing modules as well as initialize the shared variables for communication.

Although the ETP provides a high degree of flexibility for the application developer to specify the composition he wants to use for its communication, it still requires the application programmer to be "saavy". Indeed, knowledge of which Micro-Protocols to instantiate might not be trivial and requires strong expertise.

Furthermore, dynamic auto-adaptation of the transport services in response to changes of network conditions has not been provided in ETP. Indeed, even if a diversity of Micro-Protocols have been designed in the ETP framework, only the initial configuration and deployment have been considered. Dynamic adaptation strategies of the transport mechanisms in response to changes in the network service have not been defined (i.e., changes in network QoS conditions, mobility, etc).

Moreover, ETP does not provide a way for the application to detect when the service it has requested does not fulfill its QoS requirements. This results in the application having to implement its

own monitoring.

1.2.4 Conclusion on Micro-Protocol architectures

The previous sections have presented the different approaches that exist to conception of protocol architectures following a Micro-Protocol approach. These new approaches are interesting as they differ from standard ones in many aspects:

Reusability: the fact that Micro-Protocols perform only some elementary function, they can be reused without modifications in totally different assemblies to provide a totally different global service.

Configurability: because a framework is able to provide many different transport services by instantiating a composition of Micro-Protocols.

Extensibility: which allows to easily add new Micro-Protocols and instantiate them in compositions. This is made possible by the nature of the different architectures for which Micro-Protocols have to present a common interface.

Micro-Protocol composition frameworks provide a good solution to the problem of providing software components that can be adapted and composed together in order to provide adequate services to the wide variety of application requirements that exist.

However, having all the tools and pieces required to build a house does not help you if you have no idea of "how" it should be done. Indeed, you might try all possible combination of pieces together and finally succeed but by then, the potential buyers that you had when you began have already bought their house from someone else.

In order to extend the capabilities of existing composition frameworks, the following section presents different approaches to service composition either at a given or at multiple layers of the stack.

1.3 Approaches to services coordination

The existence of various services that provide either optimized or guaranteed QoS at the various layers of the communication stack have led the path for several research studies that aim at coordinating these services together. A first group of approaches considers the coordination of services executing at a given layer to provide enhanced services for one or more streams. A second family of approaches address the problem of coordinating services executed at multiple layers of the communication stack to benefit from all the enhancements while preventing undesired effects. This section presents these approaches and studies their features and limitations.

1.3.1 Mono-layer approaches

In the context of services available at a single layer, the idea is to use one or more of the available services to improve the service that is provided to applicative streams. Two families of approaches exist when considering the mono-layer scenarios, those who focus on a given data stream (mono-connection) and those who coordinate several connections together (multi-connection).

An illustration of the multi-connection proposal is the Congestion Manager approach [AAI-25] which proposes an architecture to manage congestion efficiently. Indeed, the study is based on the fact that the aggregated throughput of various TCP connections between two end-hosts is higher than the one that would be obtained by a single TCP connection on the same link. The authors propose an approach that would federate the congestion control function of the various transport connections executing between two end-hosts in order to optimize the service provided to the applications.

A similar mono-connection approach in the context of wireless networks can be found in the MULTFRC mechanism described in [M-22]. In MULTFRC several TFRC “instances” are aggregated to maximize the throughput obtained when using this congestion control mechanism on wireless networks.

1.3.2 Multi-layer approaches

This section presents coordination approaches which consider services executed at multiple layers of the stack. These approaches can either consider the full protocol stack or, in the case of modular communication protocols, the stack of Micro-Protocols in a given protocol instance.

1.3.2.1 Full communication stack

In [AAI-10], the authors address the need for real-time service provisioning at multiple layers either vertically on a single host but also horizontally across various autonomous systems.

Multi-layer coordinated adaptation is a complex task [AAI-5], indeed, it is necessary to ensure that services instantiated at one layer will not have counter productive effects on the services executing at another layer of the same stack [AAI-20, AAI-2].

When considering the global communication stack several studies have been carried on to determine the adequacy between transport level services and IP level guaranteed resource reservations. In the context of the EuQoS project, preliminary studies have shown that certain transport functions such as congestion control could lead to suboptimal performances when used on DiffServ AF classes [M-18]. In EuQoS, an extended provisioning option includes the dynamic deployment of a modular transport protocol. The Micro-Protocol stack that composes the transport instance is determined based on a statical mapping. For example, *if the IP service is Real Time Interactive, the Micro-Protocol stack should be composed of a single Shaper module shaping applicative traffic to the reserved throughput value*.

Other approaches target the context of performance optimization for small groups of cooperating peers. In this field, approaches relying on experts to define the rules to be taken into account when composing services in the form of graph grammars and graph grammar productions [AAI-14] have led to interesting results [AAI-15]. Given the assumption that the grammar and productions are correctly defined, the approach allows for coherent coordinated adaptation at multiple layers. However, this approach lacks autonomicity as the inclusion of new services requires the manual enhancement of the grammars by an expert in the field. Once the grammar has been extended, the system is able to consider the new service in its adaptation strategies.

For Micro-Protocol stacks, several approaches to improve the existing architectures for them to provide the features of an autonomic component exist. In [AAI-26], a policy based approach is introduced. The Micro-Protocol stack designer must provide a centralized adaptation manager with the policies that govern the stack's adaptation. The authors use the APPIA framework (presented in section 1.2.2.3) as a base for their implementation. By the use of policies, the service provided by the stack can be adapted to the changes in the underlying communication system's performances.

Another approach introduced by [AAI-12] consists in re-considering the layered approach by enhancing it with several mechanisms for cross-layering. In their work, the authors define the features that an autonomic Micro-Protocol stack should provide as being the automated adaptation of the structure as well as the parameters that govern the stack's execution to adapt to situation changes.

Protocol composition frameworks allow for a fine grain, highly tailored service composition to be achieved in order to compose a communication protocol that best suits the application requirements. However, the instantiation of this composition still requires the application programmer using the previously introduced frameworks to have enough knowledge on the internals of the Micro-Protocols he is composing. Autonomic Computing, presented hereafter, provides an architectural framework for overcoming these limitations.

1.4 Autonomic Computing

The previous sections have presented the wide variety of services available at the different layers of the communication stack to provide either optimized or guaranteed QoS. Additionally, Micro-Protocol composition frameworks have been introduced as a key concept for the implementation of fine-grained, highly tailored transport services for QoS. In order to coordinate the composition of these services, the previous section has presented several approaches that consider both a single layer and multiple layers.

This section presents an overview of the autonomic computing context and the main features that compose such paradigm.

1.4.1 Overview

Autonomic Computing [AAI-22] is a paradigm inspired by human observation of nature. This concept comes as a solution to the fact that applications are often required to be executed in environments that are not the ones they were designed for. To avoid a complex manual reconfiguration and adaptation of the application to adapt it to a given context, the basic properties of an autonomic component as well as an architecture to support them have been defined in [AAI-17].

The main aspect of autonomic computing resides in *self-management* which can be further refined in *self-configuration*, *self-optimization*, *self-healing* and *self-protecting*. In order to achieve this goal, an autonomic component must constantly monitor the context in which it evolves as well as the efficiency of the service it is providing. Moreover, it should be able to respond adequately to changes in the context. An important feature for an autonomic component is the ability to determine whether it can maintain an adequate level of service by adaptation or not.

Implementing an autonomic system is a great challenge. Mainly two approaches exist for doing so. The first approach consists in defining all the possible adaptations when designing the component as in [AAI-3] and [AAI-4]. This approach is suitable for systems for which all the possible adaptations can be thought of "a priori". However, for large and complex systems, this approach is often not feasible. A second approach consists in the definition of adaptations in a way that allows for their easy addition during the execution of the autonomic component. By using this technique, the actions that are undertaken by a system to adapt to its context can evolve without requiring the component to be re-engineered. The contribution presented in the following chapters is based on these concepts and benefits from these features.

1.4.2 A software designer's point of view

The autonomic computing paradigm can be represented using notation languages such as UML to reflect the implications that it has on software design. The following sections present the various elements that compose the Autonomic Computing framework from a software designer's point of view.

1.4.2.1 Autonomic Component

The autonomic computing framework features two main components which are highly related. Indeed, the basic building block of a system that follows the autonomic computing paradigm is the Autonomic Component. This can be represented as a software component which has a few basic properties such as presented on figure 1.6. Among these basic elements we find two interactions with the environment which can be materialized by software ports. One of the ports is mainly used for receiving data coming from the sensors while the other port is used to send data to the actuators. These actuators perform actions on the environment in which the autonomic component evolves.

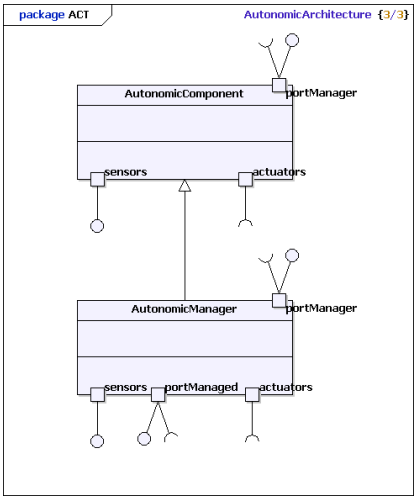


Figure 1.6: UML class diagram of an Autonomic Computing Architecture

In order to capture the hierarchical relations that govern decision making in Autonomic Computing, an Autonomic Component model also has a port through which communication with an Autonomic Manager can take place. Indeed, this port might not always be connected as Autonomic Components can operate on their own. However, the existence of this port is useful when the architecture to be implemented using Autonomic Computing paradigms requires multiple layers of decision.

1.4.2.2 Autonomic Manager

An Autonomic Manager is a specialized Autonomic Component that has all the features presented above but is also capable of managing one or more Autonomic Components. On the model, this is represented by the addition of an extra port that is used to connect the Manager with one or more autonomic components for which it will provide higher level decisions and guidelines. For example, if two autonomic components are unable to find a solution to their problem on their own, they can contact their Manager and inform it of the situation. The Manager having a higher level view of the situation (it has knowledge of the state of its various managed components) might be able to derive

local strategies that will fit the global situation and force their application on each of the managed components that experienced the problem.

1.4.2.3 Hierarchy and Delegation in Autonomic Computing

The concept of Autonomic Manager is the base to hierarchy and delegation in the autonomic computing framework. Autonomic components are located at the lowest level of the decision pyramid. When an Autonomic Component is unable to find a local solution to a given situation, it can refer to its Manager in order for it to consider the problem with a broader view of the global architecture. If, in turn, the manager is unable to find a solution, it might refer to a higher level Autonomic Manager which manages a set of similar level managers for a solution to be computed.

This architectural principle is known as delegation. Instead of always looking for a global solution to a given problem at a very high level (using much computing power to derive it), the decision process is delegated to a chain of managers and components each having a view which is more local. This principle ensures that a solution is first searched for as close as possible to the source of the problem. On the contrary, a global solution which might require higher computational power to be derived is only looked in last resort. Figure 1.7 summarizes these concepts.

1.4.3 Focus on autonomic communication

When applied to computer communication, the autonomic computing paradigm suggests that the different components of a network should perform in a way that provides a high level of self-sufficiency. In what follows, the approaches used to achieve such behavior both for the network management aspects as well as the protocol stacks' composition are presented.

1.4.3.1 Autonomic network management

For network management, [AAI-27] proposes an autonomic approach to composing the different resource reservation systems for end to end QoS guarantees across multiple Internet domains. A more radical approach is proposed by the ANA project [AAI-18] which aims at developing an architecture to support autonomic networking where protocols are not fixed but evolutive and numerous. The case

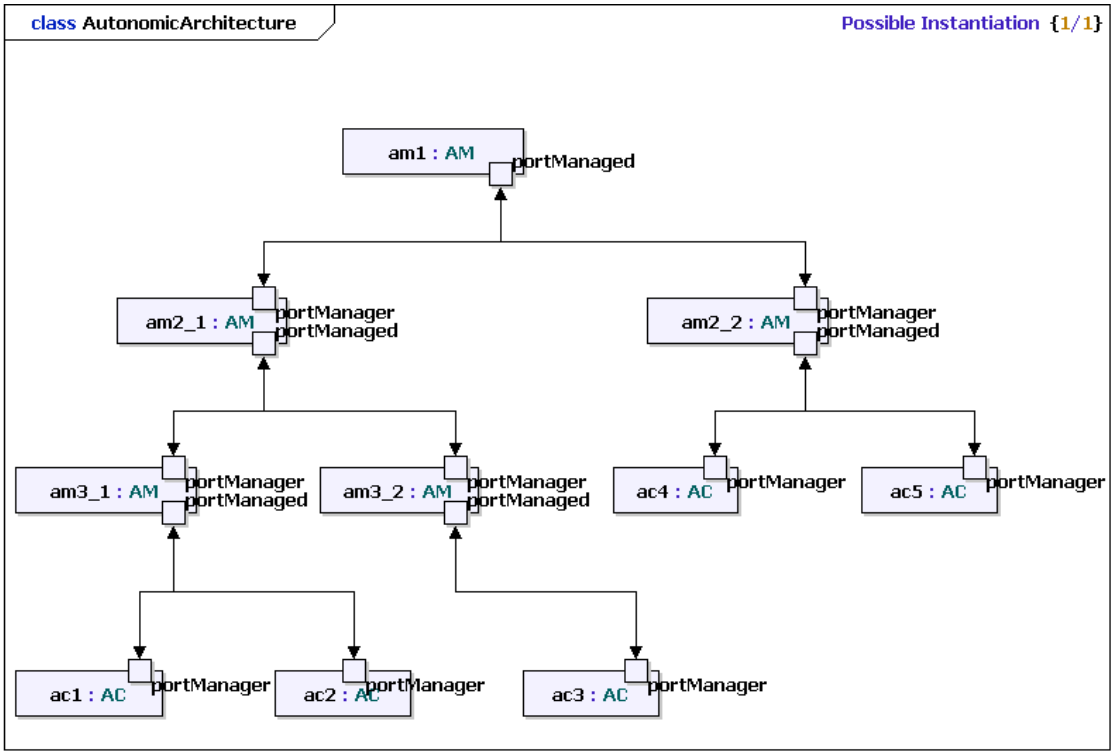


Figure 1.7: Possible instantiation of an Autonomic Computing Architecture

of autonomic multimedia communication is specifically targeted by [AAI-13] in which the authors state that UDP traffic now accounts for 25% of all Internet traffic volume (by packet count) and that autonomic network is the key to solving the problem of multimedia networks administration.

1.5 Definitions of concepts used in the manuscript

The previous paragraphs have presented several approaches to provide either optimized or guaranteed quality of service at various layer of the protocol stack. These services were either implemented on end-hosts exclusively or used equipment throughout the network to support communications.

In this manuscript, an architecture to coordinate the instantiation of these services to provide adequate communication channels to applications is presented. In the description of this architecture and its evaluation, several terms are used with specific meanings. The following sections define these notions.

1.5.1 Communication Channel

In this manuscript, an autonomic architecture to provide applications with communication channels that most closely meet their requirements is presented.

A communication channel is the generalization of what is commonly known as a socket in network programming. Indeed, a socket is highly linked to a given transport protocol. It is of common practice to refer to a TCP socket or DCCP socket.

The autonomic architecture described in the following paragraphs does not follow the same model as the application no longer chooses the transport it wants to use but instead, lets the system chose the one that is best suited based on the communication goals the application has set.

A communication channel could therefore be seen as an ACT socket as it provides the application with an endpoint for its communication through which it can send and receive data. This data will then transit through the service Composition chosen by the ACT based on the application's requirements as well as the current network conditions.

1.5.2 Service

A service is a software component that, when instantiated, provides its instantiator with communication functionalities such as sending and receiving data. The inner mechanisms of this building block are designed to address specific characteristics of the communication that will be performed by using the service. As such, a service can be seen as a generalization of the ISO definition of the notion which allows a broader range of software components than protocol layers to provide them.

For example, the EuQoS system presented in section 1.1.2.1 provides a single point of entry to the application. Once the application has instantiated the EuQoS service, the data it sends through the EuQoS service access point will be handled by inner mechanisms which aim at guaranteeing some characteristics for the data flow.

Another example of a service is a Micro-Protocol in the context of a dynamically configurable communication protocol. Indeed, when instantiating a SACK-based fully reliable error control Micro-Protocol, the instantiator's communications are provided with specific characteristics such as no losses but unbounded increase in the delay.

1.5.3 Service Composition

A service Composition is the result of the autonomic decision process. In this process, application goals are compared to the system's experience acquired through previous executions. The goal is to decide on a set of services that, when combined together, will provide a communication channel through which the data it submits will be delivered in a way that, at least fulfills the requirements if it does not outstand them.

For example, a service Composition could contain the Enhanced Transport Protocol instantiated with a TFRC congestion control [M-13] Micro-Protocol at the Transport Layer executed on top of the service provided by the NetQoS system.

Conclusion

This chapter has presented the state of the art on the different existing research fields which constitute the *building blocks* for the contributions detailed in the following chapters. This section summarizes the key concepts that have been discussed and reinforces the motivations and logic for the contribution presented hereafter.

In the previous sections, many mechanisms that provide optimized QoS at both the Network and Transport Layer have been presented. The existence of Micro-Protocol composition framework further extends the possibilities to create fine-grained highly tailored services that can be established on demand in order to best match the application requirements. Moreover, the possibility to obtain guaranteed quality through the use of frameworks such as the EuQoS system further extend the possibilities.

Using these services, an application could for example, use a stack composed of the ETP transport protocol implementing a fully reliable error control on top of a EuQoS reservation. However, this might not be the only solution which fulfills the application requirements. Indeed, the network environment characteristics might be suitable for a simple TCP/IP service over Best Effort to bring similar performance.

This example illustrates the need for coordinating the deployment of services at the various layers

of the communication stack. This coordination must take into account both the requirements of the application and the current performances of the networking environment.

The existing proposals addressing the coordination of services instantiated at either a single layer or multiple layers of the communication stack have been illustrated. However, these solutions have shown to be limited by their specificity as well as their lack of extensibility. A general observation regarding these approaches consists in stating that they generally require much information to be provided by an "expert". Moreover, they do not account for this information being approximate or false.

In order to overcome these limitations in coordinating and controlling the various existing solutions, the architectural principles of the Autonomic Computing framework have been introduced. The hierarchical architecture of the framework provides a base for autonomy by implementing various layers of *decision*. In this logic, problems are always considered for resolution at the lowest possible level to avoid unnecessary processing.

Finally, definitions of the notions to be used in the rest of this manuscript have been introduced to improve the clarity and limit the cognitive bias that could lead to misinterpretation. These notions are used in the next chapters which present the design, implementation and evaluation of the ACT. The ACT provides both an architecture and models to build an extensible system which requires little or no information from the "experts" to compose services provided at various layers to best fulfill application goals in terms of QoS.

CHAPTER 2

ACT: DESIGN OF THE AUTONOMIC COMMUNICATION TOOLKIT

A designer who is not also a couturier, who hasn't learned the most refined mysteries of physically creating his models, is like a sculptor who gives his drawings to another man, an artisan, to accomplish...

Yves Saint Laurent

Introduction

THE state of the art presented in the previous chapter illustrates the existence of several techniques and frameworks to provide enhanced QoS at multiple layers. Additionally, the need for a coherent architecture to effectively coordinate the various services available at each layer has been motivated. The limitations of existing solutions both in terms of high complexity and lack of extensibility raises the need for revising the approach. The autonomic computing framework illustrates an architectural framework for implementing scalable and fast converging systems. These principles and guidelines lay the foundations for the contributions now presented.

A first part of this chapter details the different phases of the design of the Autonomic Communication Toolkit (ACT). This design follows the Agile Unified Process (AUP), a variation of the Rational Unified Software Development Process (RUP) specifically tailored for agile methods.

The different services to be provided by the system are identified and described. The design of the

Agile Unified Process

The Agile Unified Process is a simplified version of the well know Rational Unified Process for software design. The AUP is adapted to the agile development practices such as SCRUM and eXtreme Programming (XP) in which the objective is to bring dynamism and adaptability to the process. The Unified Process for agile programming contains modified guidelines to integrate the agile $((Model/Implement/Test/Deploy)^n, Release)^m$ development philosophy into the method.

ACT uses UML 2.1 diagrams for representing the basic course of events as well as the architecture. Starting from the user and application preferences, three decision levels are identified respectively: Machine, Application and Communication Channel.

A second part of this chapter studies the problem of decision focusing on the Communication Channel level. To reduce the amount of information required from experts as well as increase extensibility, artificial intelligence theories for decision and learning are presented. For this, the Markov Decision Process (MDP) formalism is extended to take specific characteristics of the problem into account.

Algorithms for dynamically learning which service compositions provide best results are given and their extensibility is discussed regarding both new network environments and new available services. A conclusion summarizes the main aspects of the contribution.

2.1 Services Identification

The ACT is a system that provides applications with end-to-end communication channels. These channels are similar to the ones materialized by the Berkeley sockets in their usage and interface but provide enhanced services to the applications using them.

In order to provide these enhanced services to applications, the ACT uses the different services available on the machine and composes them when possible in order to conform to the applicative

requirements. These requirements are specified when the application requests the communication channel but can be changed at anytime throughout the life of the data stream. The applications can either let the system make all the decisions regarding the service to be provided, or explicitly *preempt* the system's *intelligence* by specifying the required service composition.

In addition to providing communication channels to applications, the ACT can also be used to coordinate the network resource usage of the different applications executing on a given machine. This coordination can either be regulated by the end user who communicates his preferences in terms of application's relative priorities or in an autonomous fashion by an *intelligent* component of the architecture.

Finally, several of today's applications integrate a notion of "groups" where several instances take part in a given activity. The ACT provides the architecture for group of applications executing on different machines to be coordinated to improve the overall requirements satisfaction.

The above paragraphs reveal various levels of abstraction at which the ACT has to provide services. The following sections detail the services provided at each of these levels and illustrate their use in a simple example.

2.1.1 Application wide

The first level at which the ACT services can be identified is the application level. Indeed, the ACT being a communication system, applications are the primary *actors* to which the services are to be provided.

To keep the design simple, yet providing powerful services, the use cases have been kept as close as possible to the use cases that are provided by the popular Berkeley sockets API. However, the basic socket creation and management services have been extended to provide enhanced functionalities. The resulting use cases are presented on figure 2.1, the internal component *Autonomic Service Manager (ASM)* represented on the diagram will be further detailed in section 2.2.1.1.

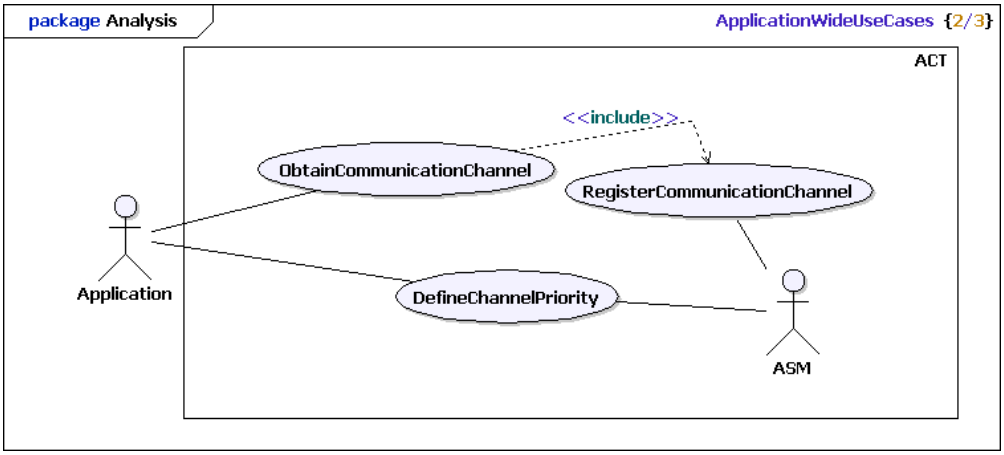


Figure 2.1: ACT Use Cases Diagram for Applications

2.1.1.1 Obtaining a Communication Channel

The basic service provided by the ACT is obtaining a communication channel. In this use case, an application requests an ACT socket from the system. The ACT socket will be the only interface between the application and the system and all management will be performed with it.

To create a Socket, the Application is requested to specify a combination of *classical* parameters such as the *source address, source port, destination address, destination port* (some of which can be omitted to let the system chose a random one (i.e., ephemeral source port selection), just like in the Berkeley socket API). In addition, the application can also specify the service composition it wants to have or let the system decide which composition is best suited by only specifying *QoS Goals*. If neither *QoS Goals* nor composition are specified, the system will offer a standard service similar to either UDP or TCP depending on the API calls that are used. The extension of the Berkeley socket API to support *QoS Goals* and compositions is represented on the sequence diagram presented in figure 2.2. On the sequence diagram, it can be seen that the *Communication Channel* constructor is highly similar to the classical *Socket* constructor. However, two additional optional parameters are taken into account in order for the application to specify its desired initial service composition as well as its QoS goals. The rest of the communication API is exactly the same as the Berkeley API, figure 2.2 illustrates datagram-based communication but ACT also provides the application with an interface for the stream-based paradigm.

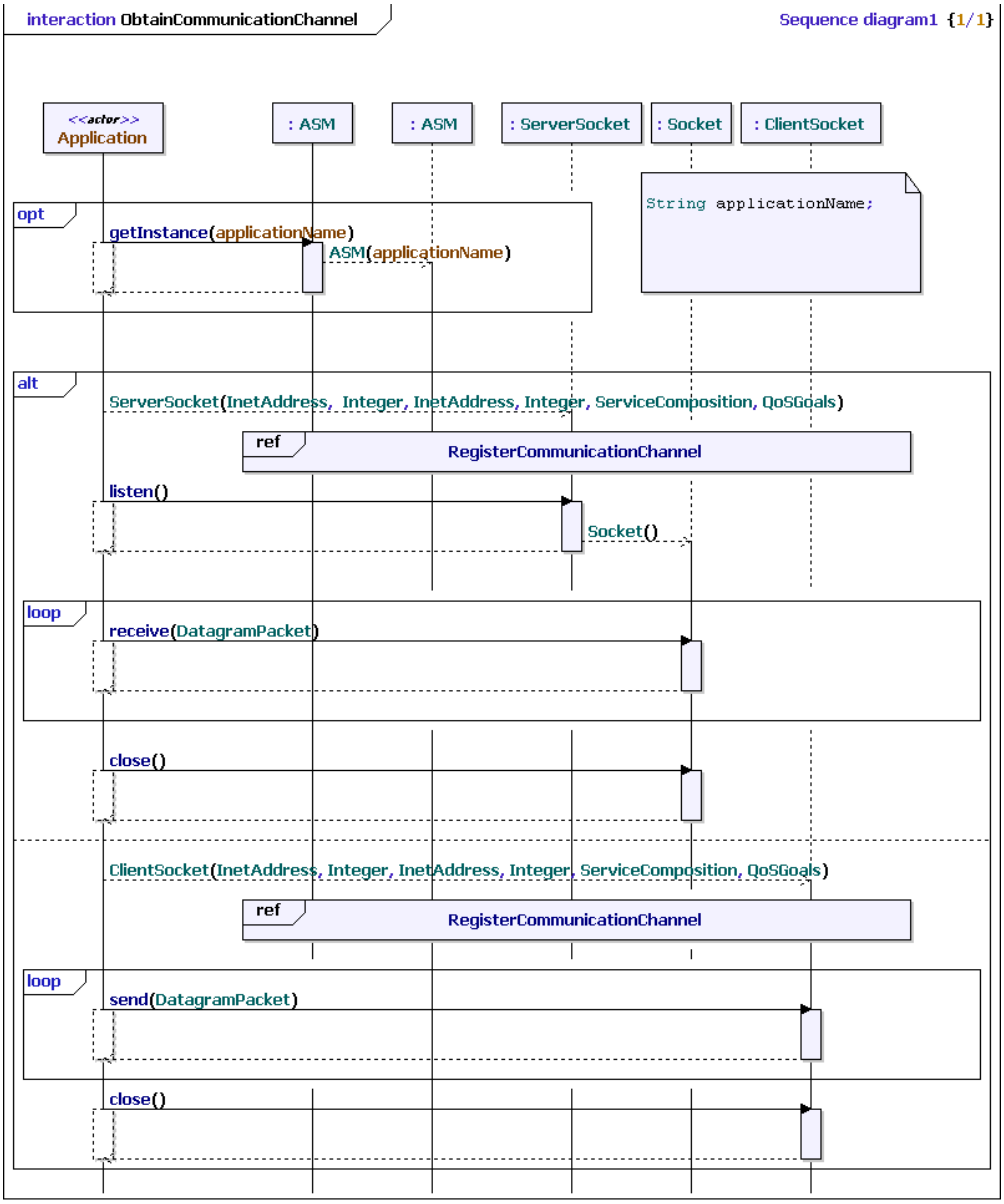


Figure 2.2: Sequence Diagram for the *Obtain Communication Channel* use case illustrating the Datagram-based interface

In the case the application allows the system to derive the best service composition based on the applicative requirements, the system will perform an initial provisioning during which all the required mechanisms are instantiated. This mechanism is transparent to the application, however, this information can be retrieved at any time and *preempted* if necessary by the application.

2.1.1.2 Defining a channel's priority

During its execution, a given application might instantiate several communication channels. For example, a multimedia transmission might require two channels for stereo audio, and a third channel for video. The network resources needed for all the streams' requirements to be fulfilled might not always be available. In these situations, the ACT will try to adapt the service composition of the different channels provided to the application. In some cases, the set of adaptation actions for each channel to lead to a global QoS goals satisfaction does not exist. However, several disjoint subsets leading to satisfying only some channels' requirements might exist.

In these scenarios, the ACT needs to be provided with additional input from the application to decide which channels to adapt service for. This additional input is materialized by a priority that the application might set for each of the channels it obtains from the ACT.

2.1.2 Machine wide

One level above the applications, several services provided by the ACT involve coordinating multiple streams generated by different applications on the same machine. Indeed, the main goal of the ACT is to provide applications with services which best suit their QoS requirements. To achieve this, the *Machine level* details the interactions that can take place among multiple streams on the same machine. The inputs for guiding these interactions is provided by the *System User* who uses the applications for communication.

These interactions are illustrated in figure 2.3, the internal components *Autonomic Service Manager (ASM)* and *Machine Manager (MM)* represented on the diagram will be further detailed in sections 2.2.1.1 and 2.2.1.2.

2.1.2.1 Monitor communications

As an autonomic system, the ACT takes decisions and performs certain actions by itself. These actions can, for example, adapt the initial service provisioning due to a change in application requirements or a change in network conditions.

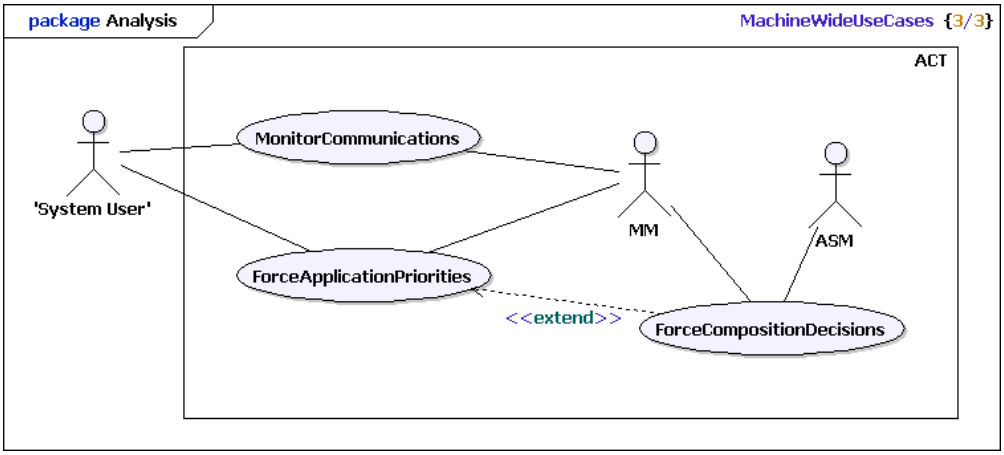


Figure 2.3: Machine Wide ACT Use Cases Diagram

To be able to observe the system's actions in response to the user expressed preferences and priorities, it is important to be able to obtain an overview of the communications that the system is managing as well as the decisions that have been taken for each communication channel.

2.1.2.2 Force applications priorities

In addition to monitoring, the user should be able to tell the system when the observed performances do not meet his expectations. Indeed, the system might be trying to reach a state where all the application stream's goals are fulfilled (which might not be possible given the current network conditions) while only a subset of these streams are of interest to the user at a given time.

In these situations, the user should be able to force the system to a certain behavior by manually specifying the relative priority of the different applications that are running on the host. This might lead the system to review its strategy and adapt its service compositions accordingly (this is represented by the *ForceCompositionDecisions* use case).

2.1.3 Summary of services

The previous sections have presented the different services provided by the ACT. A global view of these services can be found on the use case diagram presented on figure 2.4. To illustrate the system's use, a simple example is given in the box that follows.

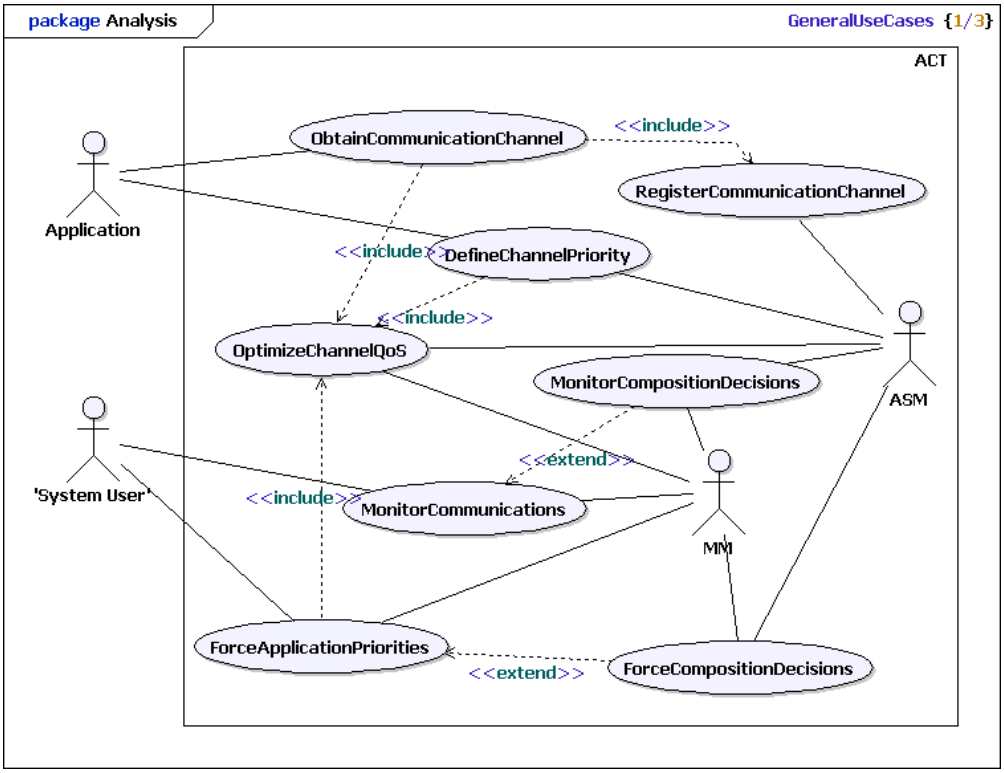


Figure 2.4: ACT General Use Case Diagram

As it can be seen in figure 2.4 presenting the use case diagram, the ACT is targeted at both applications and end users. It allows for the communications of both actors to be performed in an optimized fashion.

Several use cases which are not directly available to the external actors exist and have not been detailed yet. Indeed, the relationships between the use cases and the internal actors will be further illustrated in the next sections when the main components' design will be presented.

From the use case diagram, it can be seen that one of the central use cases of the system is to optimize the channels QoS. Indeed, this is performed through the use and collaboration of multiple actors as well as internal components using an architecture and algorithms which will be further presented in the next sections.

ACT usage example

Jack turns on his computer to listen to his favorite online radio station which provides listening software (JukeBox) based on the ACT. JukeBox requests three different communication channels from the ACT, the first one for the audio stream (high priority), a second one for a video commercial that the application displays (medium priority) and a third one for the stream control information such as artist, upcoming tunes, etc... (low priority).

A while later, Jack launches another ACT based software called BitBox Downloader which is a file sharing application.

As Jack starts downloading files from BitBox, he sees the control information on JukeBox not being updated anymore and the video commercials changing at a slower pace. Then, just as he thinks he hears a small glitch in the audio stream, he sees the speed of his file transfer reduce automatically.

A quick look to the ACT management console shows him that the system has decided to limit the service provided to BitBox to keep the JukeBox playing. As this is not what he wants, Jack changes the priority of BitBox so his transfers resume at a faster speed. The music starts getting choppy, the ACT management console shows information that JukeBox's connections have deg raded and that something could be done (if needed) to effectively exchange service from BitBox to JukeBox. Jack decides not to change the allocation.

2.2 System elements design

The previous sections have presented the analysis of the system's use cases through the identification of the different services that the ACT should provide to the various actors that use it. Although the main goal of the ACT is to optimize the QoS perceived by the end user and experienced by the applications for all the available connections, it has been seen that priorities were introduced at both the application as well as the machine level to guide the adaptation when the available resources cannot fulfill all requirements simultaneously.

In this section, the UML 2.0 based methodology is further applied to identify the different components of the ACT as well as each of these component's internal structure. This structure will allow for the system to provide all the previously identified service while keeping the possibility to extend

the system for further inclusion of novel services.

2.2.1 Architectural elements in the autonomic computing framework

The analysis conducted in the previous sections denotes two levels of abstraction for the system. A first level deals with QoS optimization in a local fashion. This is acheived by adapting the communication channels provided to an application and possibly coordinating this adaptation among the different channels of one single application.

On a higher level, the analysis revealed another form of QoS optimization where the adaptation is no longer local to a given application but can potentially lead to a coordination of various communication channels belonging to different applications executing on the same machine. (i.e., in the case of BitBox and JukeBox in example provided in the boxed paragraph)

Finally, the highest level of action has been identified as the one regarding the cooperation of various communication channels executing on different hosts executing applications that take part in a collaborative activity.

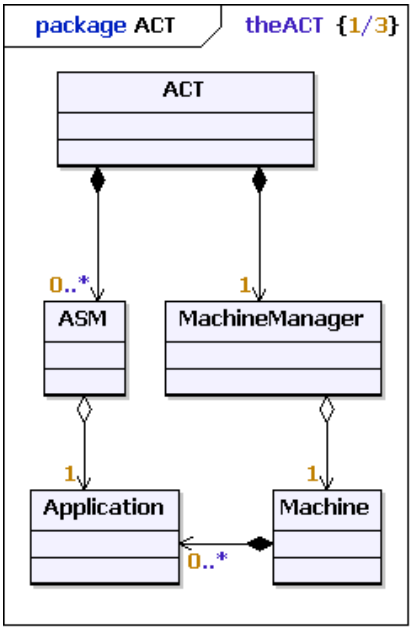


Figure 2.5: ACT Main Components

In order to take these multiple levels into account, the structure presented on the class diagram of figure 2.5 is proposed. On this diagram, it can be seen that the ACT is composed of a multitude of *Autonomic Service Manager* (ASM) components, each related to a given *Application*. Additionally, the ACT provides another component, the *Machine Manager* (MachineManager) which is related to a single machine and is responsible for Machine level services.

Additionally, the different levels presented previously introduce the notion of delegation through hierarchy. Indeed, the Machine level is only required to take action when the Application level has not been able to find a suitable solution that satisfies the applicative requirements. This allows for an

autonomic architecture to be instantiated where the Machine level will be the manager of several autonomic components at the application level which in turn will manage several potentially autonomic services operating on the data that is being sent or received through a communication channel.

Figure 2.6 presents the components of the ACT in an autonomic architecture. Indeed, the Machine Manager and the Autonomic Service Manager can be seen as specialized Autonomic Managers, the Machine Manager being itself an Autonomic Manager responsible for the various ASMs executing on the machine. These ASMs are Autonomic Components regarding the multiple services available to the applications which they coordinate together in compositions.

The structure of the main internal components of the ACT and their various interactions to provide the previously presented services are detailed in the next paragraphs.

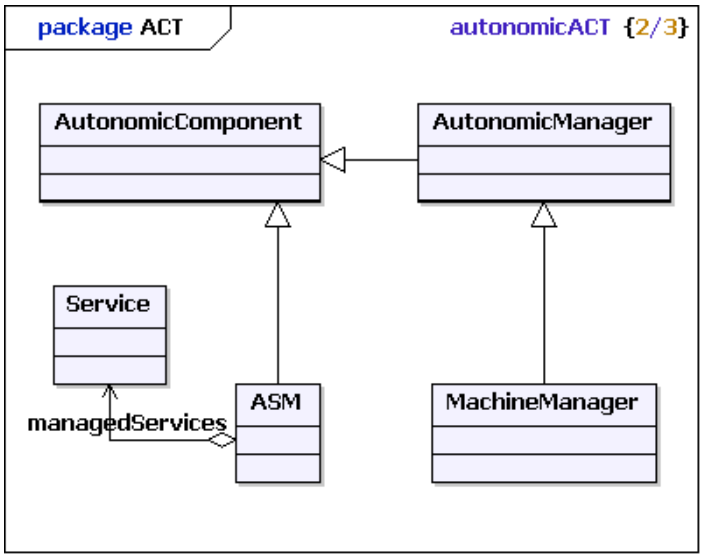


Figure 2.6: The ACT Components in an Autonomic Architecture

2.2.1.1 Autonomic Service Manager

The Autonomic Service Manager (ASM) is the ACT component operating at the Application level. Several instances of ASM might exist on a given machine but only one ASM can be instantiated for each application. It is this ASM who will be in charge of performing the various application level services such as providing the application with the various communication channels it requires as well

as making sure that these channels are adapted in a coordinated way in order to maintain an adequate level of requirements satisfaction.

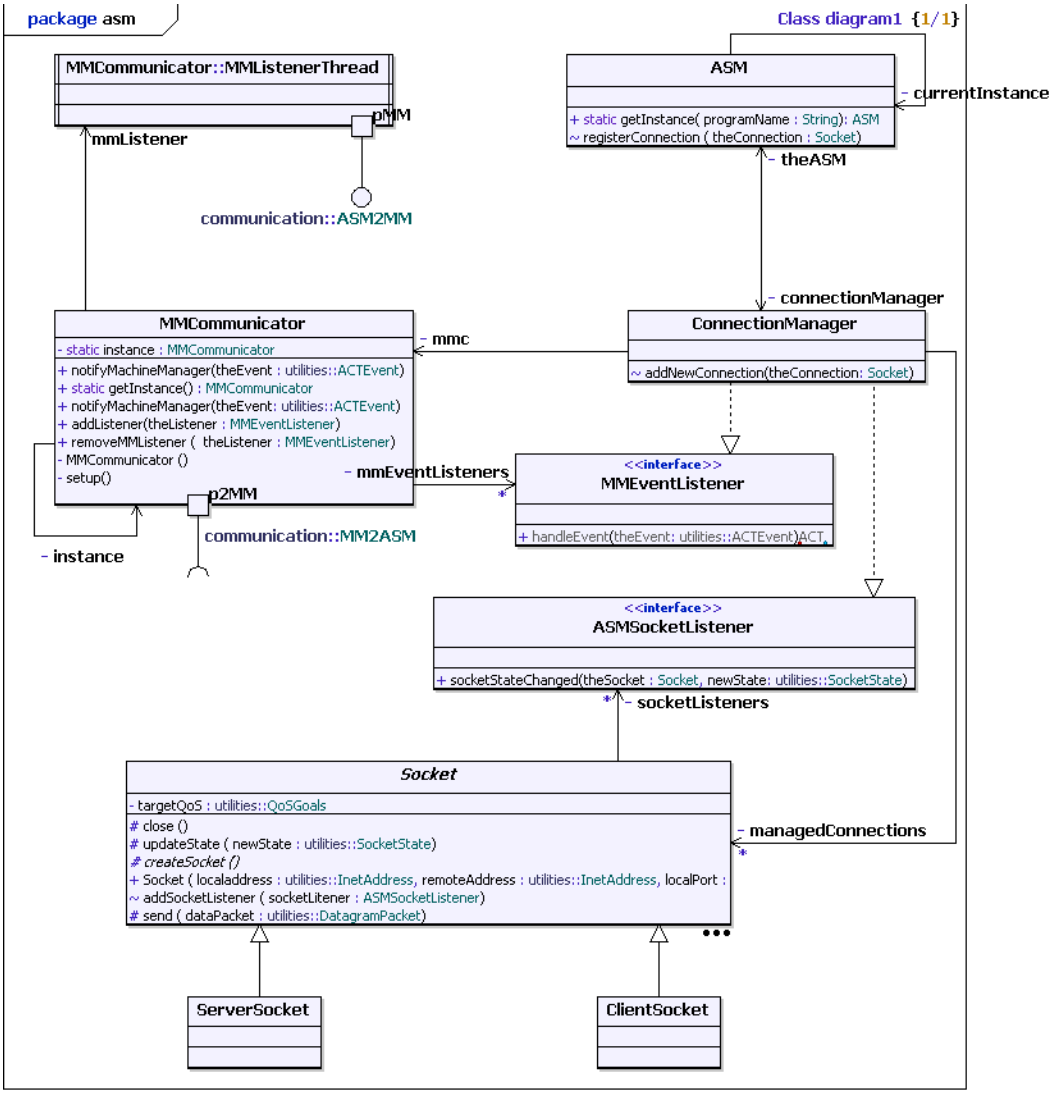


Figure 2.7: Class Diagram for the Autonomic Service Manager (ASM)

Figure 2.7 presents the class diagram for the ASM class and its various internal components. The ASM itself is materialized by a class which follows the singleton design pattern (see boxed paragraph on page 43 for details), guaranteeing that only one instance of the class exists for a given application.

The main object that applications use to interact with the ACT is the *Socket*, or one of its specialized versions (ServerSocket and ClientSocket). Although figure 2.2 only illustrates the interface for

Singleton Design Pattern

The singleton design pattern is a commonly used design principle for ensuring that a class is only instantiated a certain number of times (generally one) during the program's execution. This pattern is generally used when requiring a single instance to be globally accessible during a program's execution. The singleton's advantage over the statically defined class and class attributes is the ability to optimize memory management and benefit from dynamic instantiation.

Datagram based communication, these classes provide both the interfaces of the *Datagram Socket* which is commonly used by applications executing on top of UDP and the *Stream Socket* which is generally associated to applications using TCP.

Each time an application requests to be provided a communication channel, the application instantiates an ACT Socket by specifying its QoS requirements and optionally its preferred initial service composition. Otherwise (in absence of both composition and QoS requirements) the standard BSD Socket service will be provided. Internally to the ACT, each Socket is registered upon connection in a list contained in the ConnectionManager class. This list will be used by the reasoning algorithms detailed in the following sections.

ACT Sockets implement the *Observer Design Pattern* (see boxed paragraph on page 44 for details). This allows the design to be extensible as new components that will be used to extend the architecture will be able to subscribe to the Socket events and thus "observe" the object's changes. Although only the ConnectionManager class is an Observer to the ACT Sockets in the present design, the addition of new functionalities that require new observers to be defined is relatively easy. The ConnectionManager registers as an Observer to each newly created ACT Socket for its local (application centric) view of the system's components state to be maintained coherent. The ConnectionManager is notified of each of the events that affect a Socket's life. These events can for example be alerts or notifications related to the satisfaction of this communication channel's requirements.

The ASM internal components are sometimes required to process commands issued by the Ma-

chine Manager as well as forward relevant information to it. This follows the Autonomic Managed Component philosophy presented previously. These functions are assured by the *MMCommunicator* class. Indeed, this is the class responsible for establishing a connection with the *Machine Manager* and providing the methods for transmitting events to it. As there is no need for having multiple instances of the *MMCommunicator* class, this component's implementation also follows the *Singleton* design pattern.

The *MMCommunicator* class provides services for two way communication with the Machine Manager. To implement this, the sending of events is performed by the MMCommunicator itself in a syncrhoneous way whereas the reception (which is asynchroneous) is performed by an inner threaded class (the *MMCommunicator::MMListenerThread*). Notification of remote events are designed by using the *Observer* pattern where the *MMCommunicator* is the Subject being observed by the ASM components interested in receiving notifications from the Machine Manager. These *Observers* will then implement the *MMEventListener* interface.

2.2.1.2 Machine Manager

The previous sections detailed the internal architecture of the ASM which is the component of the ACT that represents the Application Level manager. In this section, the internal structure of the Machine level's manager of the ACT is presented in detail. The Machine Manager is responsible for managing the coordination of the different ASMs executing on a given machine. It is also responsible for higher level coordination among multiple hosts at the Cooperation Level.

Observer Design Pattern

The Observer design pattern is a software design schema used to allow an object to maintain a list of its Observers. These Observers will have to be notified when the object's state changes. This is achieved by calling one of their methods, generally defined in an interface which the Observers implement. The object is said to be an Observable Subject.

Inversion of Control Design Pattern

The Inversion of Control design pattern is a principle where the code that controls the behavior of a given class is externalized to increase the construction's reusability. Indeed, the code that controls the execution of the different tasks to be realized is written in a generic way and can therefore be easily extended to any desired control sequence.

The Machine Manager interacts with two actors. The first one is the *System User* which can perform certain tasks related to the monitoring of the different states in the ACT. The User can also perform *preemption* actions by fine tuning the priority parameter among the different applications that exist on the system. The second actor which interacts with the Machine Manager is an internal one. Indeed, the different ASMs executing on the machine eventually send notifications and reporting data about their performances to the Machine Manager.

The class diagram presented on figure 2.8 follows this distinction between the different actors which interact with the Machine Manager. The Main class is presented in the center, the initialization, routine maintenance and termination management functions are delegated to a specialized class (*MMExecutionManager*) following the *Inversion of Control* (IoC) design pattern (see boxed paragraph for details). This construction allows for easier isolation of the execution management from the core tasks of the Machine Manager as well as enhanced extensibility.

The top right part of the class diagram presents the *User Interface* of the system. The *Command-Interface* could be implemented using the *telnet* protocol for user facilitating human interaction. The menus for this command interface are defined by a class hierarchy that implements the *ICLIMenu* interface and through which navigation can be performed using language specific *reflexivity and introspection* features.

The bottom left part of the diagram presents the interface of the Machine Manager with the different ASMs executing on the host. These interactions are performed by using the *ASMListener* active class. For each ASM, an instance of the *ASMCommunicator* class is created to manage this particular

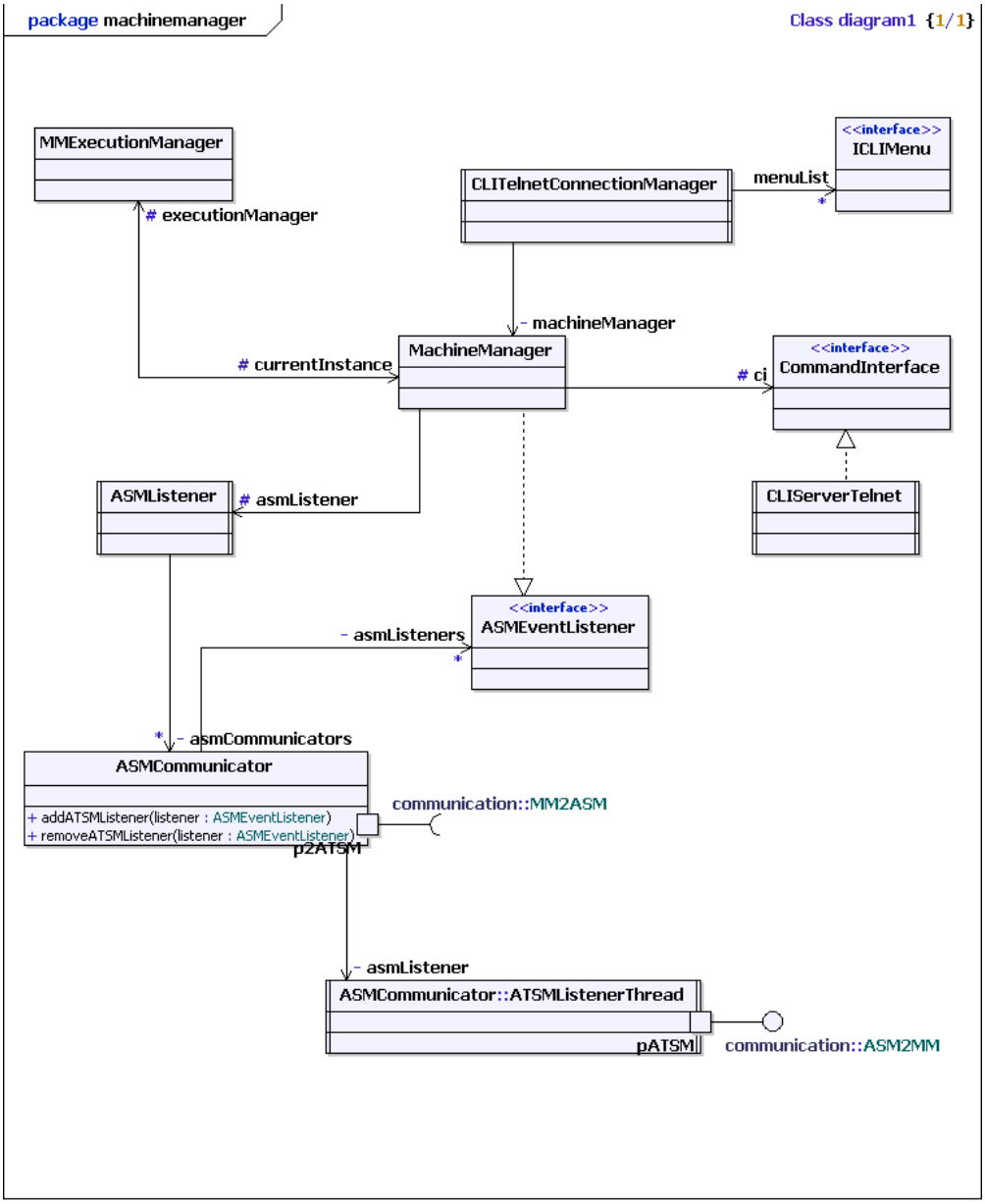


Figure 2.8: Class Diagram for the Machine Manager (MM)

communication channel. As for the ASM internal structure presented above, the outgoing communication is managed directly by this class while the incoming communication is handled through an internal class implemented as a Thread, the *ASMListenerThread*. Other internal components of the Machine Manager which are interested in being notified of events sent by the various ASMs can subscribe to them through the use of the *Observer* design pattern in a way that is similar to what has been

illustrated while describing the ASM's design.

2.2.2 Decision elements

The previous sections have presented the internal structure of the Machine and Application level components of the ACT. The architecture's goal is to provide a framework inspired by the autonomic computing paradigm and model that enables an intelligent component to obtain the information that is required for it to reason as well as the methods for it to modify the system's behavior once it has taken a decision on *what to do*.

The following paragraphs identify the decision elements in both previously presented ACT components. For each of these decision components, a discussion on the information required for the *reasoning* algorithm to be executed is introduced.

2.2.2.1 Application level decision

At the application level; the ASM is responsible for determining the best *service Composition* to be instantiated for each of the application's communication channels, unless it has been otherwise instructed by its manager (the Machine Manager). To achieve this, it is necessary for the ASM to be informed of the QoS requirements for each stream the application is transmitting. Additionally, the ASM need to be able to determine whether a given *service Composition* satisfies these requirements or not. Given these two informations, the ASM is able to determine whether an application stream requires to be adapted or not. The algorithm for deciding which adaptation action to execute will be discussed later in section 2.3.3.

In the ASM architecture presented in section 2.2.1.1 above, it can be seen that the *Connection Manager* element collects knowledge of all the different communication channels that the application has requested. Moreover, this component also implements the *ASMSocketListener* interface and can be notified, for every ACT Socket the application creates, of the different events affecting the channel. Additionally, it also implements the *MMEventListener* and has a reference to the *MMCommunicator* element which allows it to be "aware" of the Machine Manager's decisions as well as contact it with alarms regarding the Application's communication channels.

The Connection Manager element appears to be a good candidate for the implementation of the decision algorithm. Indeed, it contains all the required information for the decision process to take place and is able to contact the Machine Manager in case the local adaptation process fails to provide a suitable solution to reestablish QoS for the Application's communications.

2.2.2.2 Machine level decision

At the Machine Level, the Machine Manager is responsible for guiding coordinated global adaptations when the ASMs fail to provide a local solution to the satisfaction of their application's requirements. Additionally, it must be able to display statistics and monitoring information to the system user.

In the Machine Manager architecture presented in section 2.2.1.2 above, it can be seen that the *MachineManager* class is informed of the different ASMs' events through the use of the Observer pattern. This mechanism allows this component to be informed when the ASM is unable to find a local solution through an alarm. Moreover, the *MachineManager* element is capable of querying the different ASMs executing on the machine and by such obtains detailed information on the *health* of the other Applications executing on the machine as well as their respective connections.

For the user to be able to access monitoring information about the state of the system, the *CommandInterface* implementation presented earlier might use the services provided by the *MachineManager* element in order to dynamically discover and query the various applications to obtain the information requested by the user.

Finally, the details about the life cycle of the element being externalized to the *MMExecutionManager* class, the different functionalities are well isolated among the different subcomponents.

2.2.3 Example architecture instantiation

The previous sections detailed the different elements that compose the ACT as well as the interactions between these components through collaborative messaging. In what follows, an example instantiation of the ACT is presented in order to illustrate a possible usage for the system's internal components.

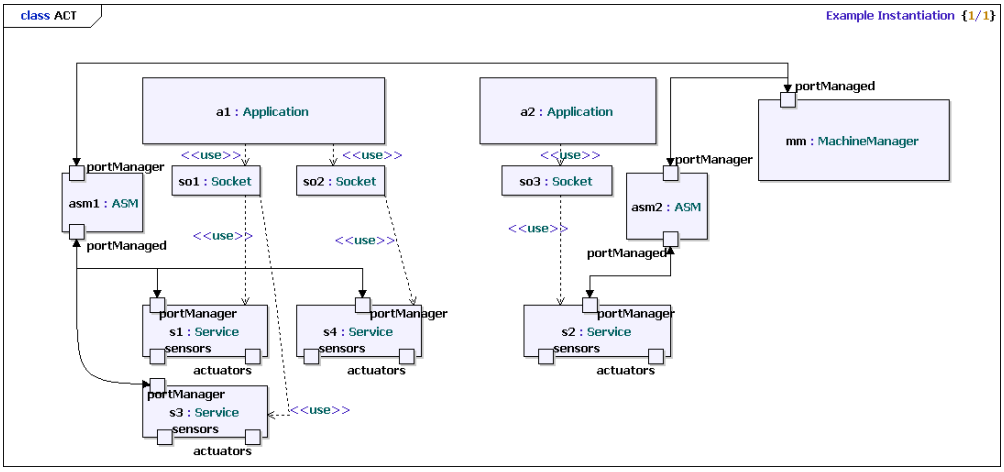


Figure 2.9: Composite Structure Diagram of a possible ACT instance

Figure 2.9 presents an example instantiation of the ACT on an end host. In this example, it can be seen that two application entities a_1 and a_2 are executed on a host and respectively make use of two (so_1, so_2) and one (so_3) ACT Socket to send and receive data on the network.

The ACT's machine level is active and a single instance of the Machine Manager (mm) is present. At the Application level, it can be seen that an instance of the ASM autonomic manager has been created for each of the two applications (asm_1 and asm_2). Each of these ASMs manages the composition of *services* that are used by the Sockets to transmit data.

The hierarchical control model of the ACT can be seen with the Machine level's Machine Manager managing the two ASMs (asm_1 and asm_2) which act at the Application level and in turn manage the different *services* which are used to support the application's communications. Additionally, it can be observed that the Application only has a single data interface to the system which is the *ACT Socket*, in a way that is totally compatible with today's most used Berkeley Socket API.

2.3 Decision strategies

The previous sections have presented a detailed overview of the main ACT components architecture. Indeed, three levels of control have been identified, the Communication Channel which composes services, the Application level which performs local control as well as the Machine level which acts as

a manager that performs global coordination. The different elements required for such an architecture to be designed have been presented.

The following paragraphs discuss the formalisms and techniques used for the design of the dynamic elements in the system. Indeed, until now only a static view of the system has been presented. The focus is made on the decision process at the communication channel level. This process gives the ACT the ability to decide on which service composition best suits specific application requirements depending on the state of the network environment. The approach that is taken consists in using techniques which are specific to the field of artificial intelligence. However, as these techniques are not specifically designed for communication systems, specific extensions that permit their use in this field are described.

2.3.1 Preliminary notions in artificial intelligence

The field of artificial intelligence finds its roots in the 1940's when part of the scientific community became interested in modelling the human brain starting by its simplest element: the neuron. This model is the basis to the now well known neural networks [AAI-16]. However, neural networks are only one of the techniques used in artificial intelligence where many other formalisms exist.

Independently of the technique used to solve a given problem, the field of artificial intelligence defines specific vocabulary to define the system. In general, a problem is described by an *agent* that evolves in an *environment*. The *agent* perceives its environment by the use of *sensors* and is able to perform actions on it through *actuators*. Generally speaking, the *agent* chooses an *action* to be performed on the environment by using the *history of its perceptions* (h). The *agent's* behavior is described by an *agent function* (f) which makes it so $action = f(h)$.

In order to define the best action to be performed by an agent, it is necessary to evaluate the efficiency of the agent's decision. This performance evaluation is specific to the problem that is being solved and needs to be defined during the agent's design. A *rational agent* will, each time a decision has to be taken, try to pick the action that will maximize its performance. The process by which the decision is taken depends on the environment's nature as well as the nature of the problem to be solved. In what follows, techniques for decision making in the presence of uncertainty as well as

learning in such environments are presented.

2.3.1.1 Reasoning in a perfect world

In a perfect world for artificial intelligent agents, the context in which they evolve is totally known to them and the effects of the different actions that they might undertake is totally predictable.

In these environments, the decision making process is generally formalized by means of an optimization problem. Indeed, the goal is to find a policy π, defined as a sequence of n actions $a_1, \ldots, a_n$ that solve a given problem while maximizing the solution's efficiency.

The general method for solving these problems is exploration of the solution space. This exploration can be either performed without prior knowledge of the problem to solve, in which case it is referred to as *uninformed exploration*. This class of methods is opposed to *informed exploration* which takes the specificity of the problem into account to find the solution faster. The following sections detail some of the methods used in both kinds of explorations.

2.3.1.1.1 Uninformed exploration In uninformed exploration of a perfect world, the agent is provided with only two elements: the current state of the environment (E_t) in which it evolves and the set of actions (A_t) that it can execute at any given time t.

Given these two elements, its goal is to find a policy $\pi = (A_t)_{i i \in [1..n]}$ that will make it so the final state of the environment (E_{final}) corresponds to the target that the agent was to achieve.

In these situations, the basic solution is to explore the complete set of possible policies to find one that will lead to E_{final}. Although this technique is simple to implement, it is not very effective as the average time required to find a solution grows exponentially with the number of states of the environment an the possible actions the agent can take.

To reduce the complexity of the problem, additional data is required to be taken into account during the decision process. This data can either be provided at design time by the designer or acquired during run time through learning techniques detailed later in this chapter. Independently of the way the additional data is acquired, the resulting problem is no longer of uninformed exploration, it becomes an informed exploration problem.

2.3.1.1.2 Informed exploration In informed exploration of a perfect world, the agent is provided with the same elements as in uninformed exploration but it also has a model of the environment in which it evolves.

The goal of informed exploration is the same as the goal in uninformed exploration which consists in finding a policy (π) that will take the agent from its initial state $E_{initial}$ to its goal state E_{final}. However, the tools for achieving such task are different.

Informed exploration can lead to much better results than uninformed exploration. This is due to the fact that the agent can use the model (M) of the environment that it is provided with to compute the effects of its actions. Indeed, at any given time t, and given the environment state E_t, the agent is able to compute the state of the environment $E_{t+1|a_i}$ resulting from the decision to apply action a_i by computing $M(E_t, a_i)$.

The world being perfectly modeled, the complexity of the problem can be reduced in the sense that all future states of the environment can be computed without actually performing any action. Moreover, the future states being predictable, the agent can be provided with a way to determine the *efficiency* of a state and compare two possible futures. This allows the agent to choose the *best* action at each decision and reduces the space of states that require exploring thus greatly reducing the complexity of the problem.

2.3.1.2 Reasoning in the presence of uncertainty

The previous sections have illustrated how artificial intelligence techniques can help computer systems solve problems in a perfect world. However, real world problems can rarely be represented by a totally deterministic model. This leads to a certain level of uncertainty in the process. The uncertainty can be of two kinds. Uncertainty can be due to the fact that the agent is unable to determine the environment's state because it depends on variables that the agent is not able to capture. Additionally, uncertainty can also arise from the fact that the agent does not know the *exact* model for the environment in which it evolves. This makes it so the outcome of the decisions taken by the agent can only be assumed with a certain degree of confidence which has to be taken into account in the decision algorithms.

When the agent is uncertain about the environment it evolves in, it means that no deterministic model of the context can be established and that the actions carried on by the agent have effects that cannot be predicted with total certainty. As opposed to reasoning in a perfect world, where the system's context can be completely modeled, the effects of an action undertaken by the agent on the environment can't be computed with certainty prior to the decision. These limitations can be modeled in order for the model to take uncertainty into account by using formalisms such as Bayesian Networks and Markov Decision Processes [AAI-24].

Additionally, when the model of the environment is unknown, it is possible to use techniques known as *artificial learning* to build it automatically. This can either be done off-line, during the agent's design phase, or on-line, once the agent is running in its production environment. The following paragraphs present both the Markov Decision Process and the reinforcement learning formalisms which are strongly related.

2.3.2 Improving accuracy by the use of learning techniques

The previous paragraphs have introduced the interest of having a model that allows for an agent to take decisions based on informed exploration. The next paragraphs introduce Markov Decision Processes, a discrete time stochastic control process which models the effects of an agent's action on the environment when these are not certain. Additionally, reinforcement learning is presented to cope with the variability of the environment and adapt to its changes in real-time.

2.3.2.1 Markov Decision Process

A *Markov Decision Process (MDP)* [AAI-24] is an extension of the *Markov Chain* formalism in order to add multiple possible actions to each state allowing for a choice to be made.

Formally, an MDP is defined by a tuple $< E, A, R, P >$ with E a set of possible states, A a set of possible actions, P a probability distribution function such that $P(e_1, e_2, a) = Pr(e_{t+1} = e_2 | e_t = e_1, a_t = a)$. $P(e_1, e_2, a)$ gives the probability that, given the modeled system is in state e_1 at time t and the agent performing action a, the system transits to state e_2 at time $t + 1$. Markov Decisions Processes are often also associated with a reward function $R(e_1, e_2)$ which gives a numerical value

corresponding to the reward an agent would get by having the system perform a transition from state e_1 to e_2 (this is mostly used in reinforcement learning applications as detailed in the following sections).

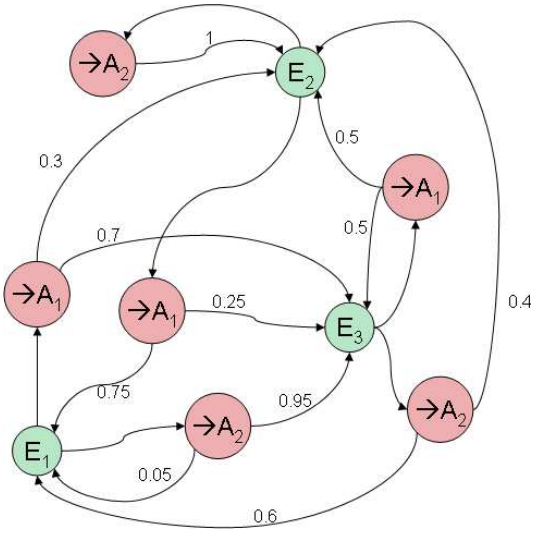


Figure 2.10: Example instance of an MDP

Figure 2.10 presents an arbitrary example of an MDP. In this instance, three different environment states (E_1, E_2, E_3) are considered in which the agent always has two possible actions it can perform (A_1 and A_2). The actions performed by the agent will impact the environment by modifying its current state. However, the resulting environment state is uncertain as captured by the probabilities displayed on the graph. For example, if an agent senses its environment in state E_2, performing action A_2 will have no impact on the environment state while performing action A_1 will most likely make the environment transit to state E_1.

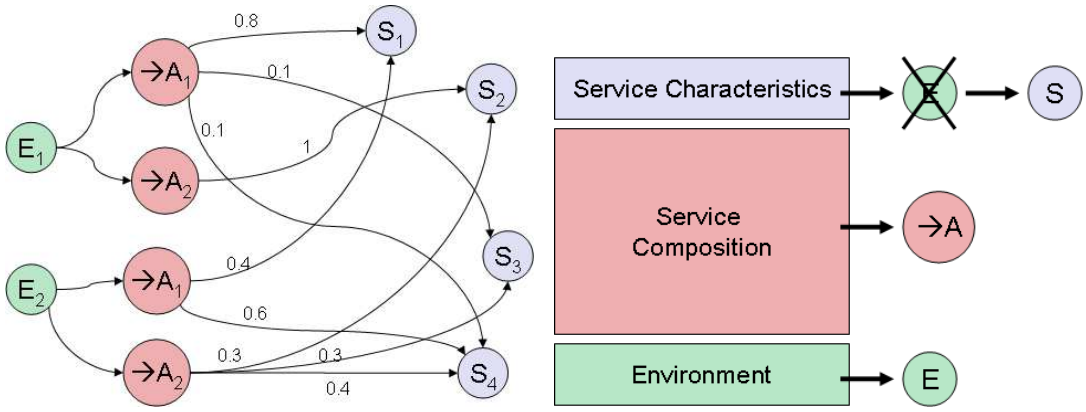
When considering the goals of an agent that must decide on the best services composition in order to provide an application with a service that best suits its requirements, the MDP does not capture all the required information. For instance, as presented on figure 2.11(b), the agent is interested in the characteristics of the service (S) obtained by applying a composition (A) in a given network environment (E). Depending on the nature of the services composed in A, S can be radically different from E (for example, S can be lossless while E is not). An MDP only captures the effects that the use of composition A might have on the environment, an MDP does not allow to capture the resulting

service characteristics. To overcome the limitations of classical MDPs, the MDP concept has to be extended as explained hereafter.

Given a current state of the environment e_t, the choice of a given composition a by the agent will provide the application with a communication channel having a set of characteristics s_k which depend only on the environment state and the composition: $s_k = f(e_t, a)$.

Let there the tuple $< E, A, S, R, P >$ where E is the set of possible states for the network environment, A is a set of possible compositions that can be applied to "improve" the service offered by the network, S is the set of service characteristics resulting from the application of elements in A to elements in E. P a probability distribution function such that $P(e_i, s_k, a) = Pr(s_{t+1} = s_k | e_t = e_i, a_t = a)$. $P(e_i, s_k, a)$ gives the probability that, given the environment is in state e_i at time t and the agent is applying composition a, the application is provided with a service having characteristics s_k at time $t + 1$.

One of the direct effects of this enhancement to the definition of the MDP is that it is now limited to first order. This greatly simplifies the decision process as well as limits the size of the representation. An example of an extended MDP instance is presented on figure 2.11(a). The graph considers two environment states (E_1, E_2) in which the agent can choose between two different service compositions (A_1, A_2). The choice of a given composition leads to four possible services (S_1, S_2, S_3, S_4) to be delivered to the application. The uncertainty in the effects of an agent's actions are captured by the



(a) Example instance of an extended MDP

(b) Relationship between E, S and A

Figure 2.11: Extended Markov Decision Process

probabilities on the graph. For example, an agent using the extended MDP presented on figure 2.11(a) for decision making and sensing the environment in state E_1 "knows" that applying composition A_1 is most likely to result in service S_1 being provided to the application.

2.3.2.2 Reinforcement Learning

The model for reinforcement learning is presented on figure 2.12, and is further detailed in [AAI-19]. In this model, the agent (P) senses the state e of the environment (E). Based on this sensed state, the learner (L) updates the knowledge represented by a model (M). Each time it is required, the controller (C) decides to execute action a which will have some effects on the environment. The effects of action a are sensed by the agent by both, the new value of the environment state e as well as a numerical value r which evaluates the decision of the agent to perform action a. This value is called the *reinforcement*.

The problem of reinforcement learning consists in discovering the best action or sequence of actions to be performed to control a system which is in an actual state e_t towards a "goal" state e_g. To achieve this, several methods have been studied in the literature and are summarized hereafter.

In the rest of this dissertation, it is assumed that the system effects on the environment can be modelled by an extended MDP $< E, S, A, R, P >$ where E is the set of states the environment can be in, S is the set of services characteristics that can be obtained by applying actions in A which is

Learning Techniques in AI

In the field of Artificial Intelligence, several techniques exist for an agent to *learn* how to control a given system. These techniques can be categorized as supervised and unsupervised. In supervised learning, several input/output pairs are presented to the agent for it to determine a *model* of the system it will supervise or control. In unsupervised learning, the agent is never presented with such data. Instead, it must determine the best behavior by successively trying the various possible actions and observing their effects.

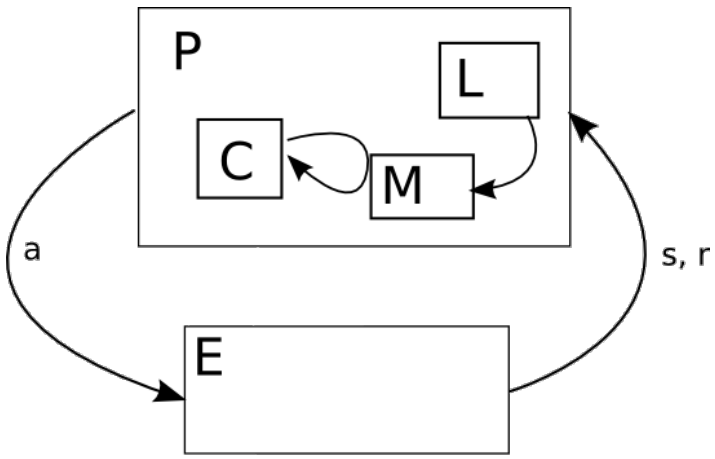


Figure 2.12: Reinforcement Learning Model

the set of possible actions available to the agent, R is the distribution of the rewards available to the agent and P represents the distribution of probabilities for each (state, action) pair in the MDP.

In the case of MDPs in the context of networking, the reward function can only be computed online. This is not a major drawback as the computational overhead induced by the decision process is relatively small as reconfigurations should not occur too often, only when the service goals are no longer met by the current composition. In the context of networking, the reward is generally not immediate. [AAI-23] studies learning from delayed reward.

The complexity of reinforcement learning relies on the fact that the agent has no prior knowledge of the underlying MDP; or in case it has some knowledge, it might be limited to a subset of the full picture.

When the environment is stationary, meaning that the transition probabilities on the MDP do not vary over time, the learning process can sometimes be performed off-line. However, this is not always feasible because the environment can not always be easily simulated (i.e., in the case of an autonomous helicopter pilot [AAI-1]) due to the great complexity of varying factors.

When the stationarity of the environment is not assured, meaning that the transition probabilities vary with time, several approaches have been considered to extend existing reinforcement learning algorithms to take these variations into account. One of them is the active reinforcement learning described in [AAI-7]. In these cases, learning can only be performed on-line during the system's

execution. The controller then takes decisions on the model that is available at time t and the effects of these decisions are taken into account to adapt the model for next decisions at time $t + 1$.

Several studies such as the one performed in [AAI-11] have shown that, providing the transition probabilities' unstationarity is bounded by a small value ϵ, simple extensions to the algorithms that provide optimal policies in stationary environments lead to near optimal policies in varying environments.

The following section describes the use of the extended MDP and reinforcement learning theories to implement the decision process that is responsible for composing available services in order to build communication channels that best suit application requirements.

2.3.3 Decision process for communication channel service compositions

When an application requests a communication Communication Channel to be created, it describes the service it is expecting to get from the ACT in the form of a set of requirements encapsulated in a *QoSParameters* class structure. The *QoSParameters* class is composed of a set of end-to-end performance elements such as maximum delay that can be tolerated (D_{max}), maximum loss rate that the data can experience (L_{max}) as well as a minimal throughput for the data (T_{min}).

Given the parameters provided by the application, the ACT must instantiate a *Protocol Stack* by selecting the adequate service at each of the layers it has control over. This initial provisioning is then enforced and monitored throughout the life of the data flow. If performances degrade below a certain threshold, the ACT might decide to re-evaluate its decision.

The following sections present the model that is used to perform these actions in an autonomic fashion as well as the techniques used to acquire the model through learning.

2.3.3.1 Introductory notations

Let there a, the state of the agent responsible for autonomically composing the stack for an applicative flow having performance goals $g = \{D_{max}, L_{max}, T_{min}\}$. In this case, a represents the current composition. Let there A, the set of possible states of this agent. A is composed of all the possible service compositions. In the case of a stack of n layers each being composed of $L_{i\in[1..n]}$ possible

services, the cardinal of A is $\prod_{i=1}^{n} L_i$ if services are all considered to be compatible with each other. This constitutes the worst case scenario in which the possible state space is the biggest.

Let there e_t, the sensed state of the network environment described by $e_t = \{D_t, L_t, T_t\}$ where D_t is the mean delay measured over the interval $[t-1, t[$, L_t and D_t are respectively the mean loss rate and the throughput measured over that same interval. Let there E the set of all possible environment states.

Let there $SA =< A, \sum, T >$ the semiautomaton that describes all the possible behaviors of the agent. In case there are no restrictions on the transitions that the system can perform, then SA can be represented as a strongly connected graph as illustrated on figure 2.13.

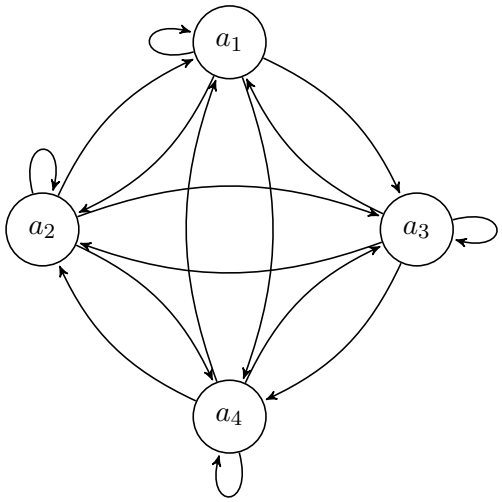


Figure 2.13: Example semiautomaton in the case of four states without restrictions

Finally, let there $\gamma : G \times S \rightarrow \mathbb{R}$ a reward function that meters the capacity of service characteristics $s \in S$ to fulfill applicative goals $g \in G$. $\gamma(g, s)$ is defined so that $\gamma(g, s) < 0$ in the case the goals are not fulfilled by s and $\gamma(g, s) \geq 0$ otherwise. Moreover, given two possible service characteristics $(s_1, s_2) \in S$ under goals g so that $\gamma(g, s_1) > 0$ and $\gamma(g, s_2) > 0$. If s_1 fulfills g better than s_2 in the sense that it keeps the system further away from the limits defined in the goals than s_2 then, $\gamma(g, s_1) > \gamma(g, s_2)$.

2.3.3.2 Knowledge representation

The decision of which action a to perform when a given service characteristics s does not fulfill the application's goals g (i.e., when $\gamma(g, s) < 0$) is not easy to make. Indeed, several approaches exist for doing so. In the previous chapter, some of the main techniques for building autonomous systems capable of taking such decisions have been presented.

Although it is possible to have a networking expert define a set of rules to govern the decision process in stack composition, this solution might be extremely costly and would only be viable for *design time* specified environments. The rules that compose the set would take the form: *when in environment state e or similar and goals are not met, the best option is to perform action a.* It appears clearly that this rule based approach is only viable for a universe composed of few possible environments and not too many actions. On the contrary, an intelligent agent should be able to adapt to new situations which were not explicitly *thought of* during design. Moreover, because the observed values of the environment parameters might not characterize it completely, the outcome of the decisions might not always be the same.

To overcome the limitations of an expert defined rule based system, one approach is to use previous experience as a knowledge representation. Indeed, the environment (in our case, the communication network) can be affected by a broad, yet limited range of events (i.e., congestion, routing problems, ...). The effect of these events have an impact on the parameters that can be observed by an agent such as *delay, bandwidth, losses*. Moreover, it is likely that these events occur several times throughout the life of the environment with comparable impact on its state or the observable parameters. Thus, it is likely (although not certain) that actions which brought good results in the past will lead to similarly good results in the future.

The knowledge of the effect the different actions an agent has on the characteristics of the service provided to application are modeled in an extended Markov Decision Process (MDP) which is built during the agent's execution through active learning techniques [AAI-7]. The MDP is defined over E, the set of possible states for the environment and the agent's reconfiguration actions which correspond to the possible transitions in the agent's semiautomaton SA defined previously given its current state $a \in A$. This is illustrated on figure 2.14 for environment (network) state e_t and actions a_k, a_t, a_m. In

this case, the decision to apply composition a_k might lead to two possible services (s^1 and s^i) being delivered to the application with probabilities $P(s^1_{t+1}|e_t, \rightarrow a_k)$ and $P(s^i_{t+1}|e_t, \rightarrow a_k)$ respectively.

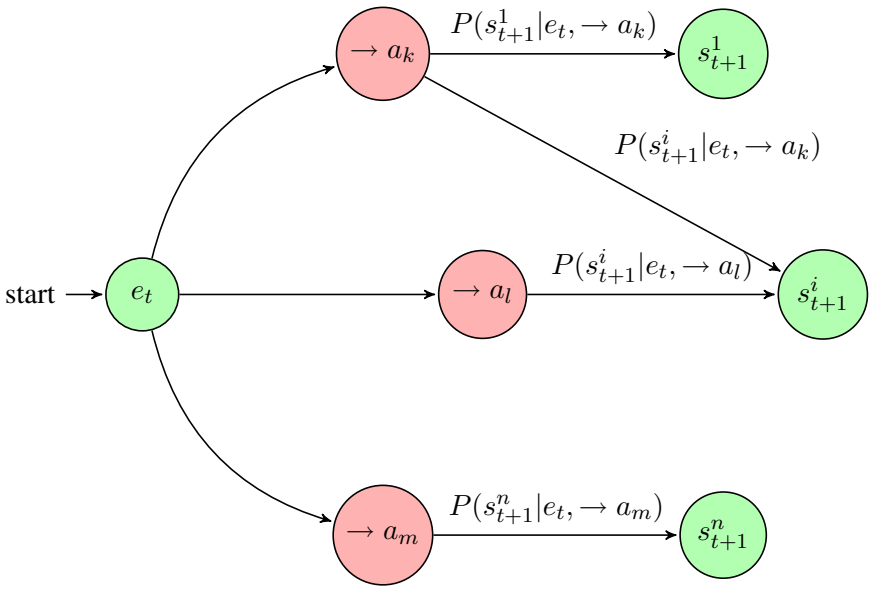


Figure 2.14: Partial representation of the knowledge as an extended MDP

2.3.3.3 Environment and service characterization

In the previous sections, the environment state as well as the characterization of the service provided to the application have been referred to several times.

The environment state is obtained by monitoring the *Throughput, Delay and Losses* parameters when using a "standard" service composition. By "standard" service composition it is understood that it is the service composition that is obtained by using the communication system that an application would get if it was not using the ACT and choosing UDP as a transport protocol (i.e., without activating any service).

The characterization of the service provided to the application is obtained by monitoring the same parameters as for the environment state (*Throughput, Delay and Losses*) over the Communication Channel resulting from the service composition that has been chosen. This monitoring can be performed in-band by encapsulating the application data in ACT packets which contain monitoring

information in their header.

For example, in the case of a service composition composed of a SACK-based (Selective Acknowledgment) fully reliable error control at the Transport Layer and Best-Effort IP at the Network Layer, the service characterization would have a null losses parameter while the environment state might have a different losses parameter value.

2.3.3.4 Decision process

The decision process for the ASM consists in choosing a service composition $a_d \in A$ that, given the user defined goals g, makes the service provided to the application $s_t \in S$ where s_t best fulfills the goals.

In this section, the assumption is made that the ASM has knowledge of both a, the current service composition and e, the current state of the environment at all times. Moreover, the semiautomaton that governs the possible state transitions on SA is also known to the ASM.

To perform the decision, the ASM requires knowledge of an extended MDP such as the one presented on figure 2.14. This MDP is built online during the agent's execution based on its past experiences. Details of the bootstrapping and learning process are provided in the following section (2.3.3.5).

Formally, each time the current characteristics of the service provided to the application do not satisfy the user goals g, the ASM should decide to execute the state transition (reconfigure the stack) that will consist in instantiating the service composition a_d which maximizes the probability for the service provided to the application to evolve into s^d_{t+1} satisfying $\gamma(g, s^d_{t+1}) = max_i\big(\gamma(g, s^i_{t+1})\big)$.

Definition of the γ function.

Let there $g = (D^g_{max}, T^g_{min}, L^g_{max})$ the applicative goals with D^g_{max} the target maximal delay, T^g_{min} the target minimal throughput and L^g_{max} the target maximal loss rate.

Let there $s = (D^s, T^s, L^s)$ the service characterization with D^s the current average delay, T^s the current average throughput and L^s the current average loss rate.

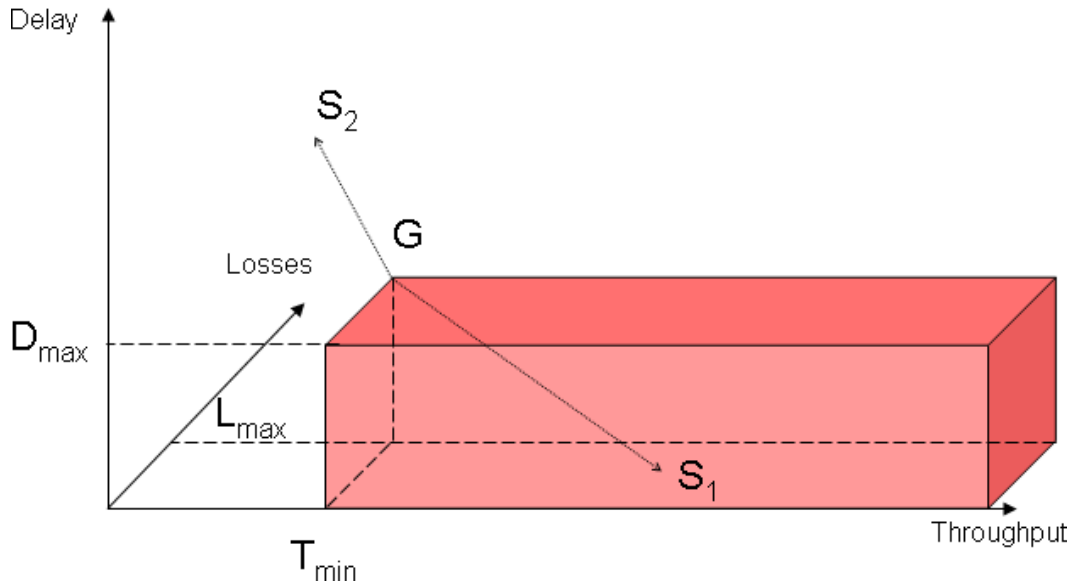


Figure 2.15: Graphical Representation of the γ function's sign

The s and g elements both have a graphical representation as points in $\mathbb{R}^3$. Let there S and G these points.

$$\gamma(g,s) = \begin{cases} \|GS\|_{L_2} & \text{if } s_d \leq g_d \text{ and } s_l \leq g_l \text{ and } s_t \geq g_t \\ -\|GS\|_{L_2} & \text{otherwise} \end{cases}$$

where $\|GS\|_{L_2} = \sqrt{(D^s - D^g_{max})^2 + (T^s - T^g_{max})^2 + (L^s - L^g_{max})^2}$; the L_2 norm.

Figure 2.15 shows a graphical representation of the γ function's sign. Inside the area defined by the box (it is the case of S_1 for example), the γ function is positive while it is negative in the rest of the space (the case of S_2).

2.3.3.5 Learning process

In the previous section, the decision algorithm allowing for the ASM to select the *best* service composition to apply to take the environment closer to the user defined goals has been presented. The decision process greatly relies on an MDP that represents a partial view of the possible futures for the environment depending on the state transitions the ASM performs.

This MDP is obtained through learning and refined each time an action is taken by the agent. Indeed, upon initialization, the MDP is totally unknown. The only thing that might be known to the ASM are the sets E and S as well as e_t and s_t characterizing both the current service composition and the environment state.

When the agent is first started, it has no knowledge of its environment whatsoever. However, it is capable of rapidly characterizing the environment in which it evolves by measuring the transit delay and packet loss rate as well as throughput. The agent is also aware of its current state, in this case a default (i.e., UDP/IP) service composition, as well as the semiautomaton that informs it of the transitions it might perform. These transitions might have unequal costs. For example, going from a UDP/IP service composition to a UDP/EuQoS composition could be assigned a higher cost than going from UDP/IP to ETP/IP as the EuQoS invocation process takes longer to complete.

2.3.3.5.1 Bootstrapping process

The first few times a decision is to be taken, the decision process described above cannot be applied directly to the problem as the MDP is unknown or incomplete. The learning process needs to be initiated. This is done by randomly choosing an action among the ones with the lowest cost in the agent's activated transitions given its current state.

Once the action is performed, the agent will start monitoring the resulting service to evaluate

Exploration vs. Exploitation trade-off

In reinforcement learning techniques, a well known problem is known as the exploration vs. exploitation trade-off.

The basic idea behind this problem resides in the fact that each time a decision has to be taken, the agent has to chose between either taking the best action among the actions it knows will lead to desired states or exploring actions that it has not yet experimented and which could lead to disastrous results.

its adequation to the application's requirements. Should the provided service satisfy the applicative goals, no further action is taken and the transition probability for the given action to provide the currently observed service is set to 100%.

The next time the agent is confronted with a need to decide on a composition to setup for the same environment state, it can either test compositions which have not been tested yet or go for the best known choice. This is known as the exploration vs. exploitation trade-off (see boxed paragraph) which is a common problem in artificial intelligence learning for which a probabilistic approach is adopted similarly to what is described in [AAI-6].

In the implementation of the ACT, the choice has been made to always explore if none of the current solutions fulfills the applicative goals (i.e., $\gamma < 0$). However, when there is one known action that will lead to an adequate service with regard to the application goals, the probability that exploration takes place decreases with the number of remaining compositions to test following the bayesian exploration technique described in [AAI-6].

2.3.3.5.2 Algorithms for decision making and learning

The general algorithm for decision making by taking the exploration phase which is required for learning into account is described in pseudo-code notation as algorithm 1.

In the decision algorithm, it can be seen that only the actions which are possible successors to the current action in the semi-automaton are taken into account. After a first glance at the actions to determine if they have already been performed on the environment in its current state, all known futures (service characterizations) that can be obtained by performing actions on e_t are revised to determine whether one of them is adequate with regard to the application goals.

The next part of the algorithm, ranging from lines 19 to 28, define the exploitation of existing knowledge by looking for the best action to be performed to obtain a service which's characterization best suits the application defined goals.

Finally, lines 29 to 31 define the exploration strategy which consists in first exploring the solution which has the lowest cost as defined in the transition probabilities on the semiautomaton.

Algorithm 1 Decision Algorithm

Require:

The current state of the agent a_t (the current composition).
The current characteristics of the service provided to the application s_t.
The semiautomaton for the agent SA.
The agent's experience H (an MDP such as the one illustrated on figure 2.14).
The current state of the environment e_t.
The application goals g.
The reference to the ASM, asm.

```
1: actions[] ← SA.getPossibleSuccessors(a_t)
2: services[] ← H.getKnownPossibleServices(e_t)
3: decision ← a_t
4: allExplored ← true
5: for all actions : a do
6:     if not H.areServicesKnown(a) then
7:         allExplored ← false
8:         toBeExplored.add(a)
9:     end if
10: end for
11: hasGoodAction ← false
12: for all services : fs do
13:     if γ(g, fs) > 0 then
14:         hasGoodAction ← true
15:     end if
16: end for
17: Pexploit ← random(0.0, 1.0, uniform);
18: tradeoff ← sizeOf(toBeExplored) / sizeOf(actions)
19: if allExplored OR (hasGoodAction AND Pexploit > tradeoff) then
20:     bestAction ← a_t
21:     bestEfficiency ← γ(g, s_t)
22:     for all services : fs do
23:         if γ(g, fs) > bestEfficiency then
24:             bestEfficiency ← γ(g, fs)
25:             bestAction ← fs.getBestAction()
26:         end if
27:     end for
28:     decision ← bestAction
29: else
30:     decision ← getLowestCost(toBeExplored)
31: end if
    asm.performAction(decision)
```

Algorithm 2 Learning Algorithm

Require:
The current state of the agent a_t.
The agent's experience H.
The application goals g.
The reference to the ASM, asm.

```
1: loop
2:     currentE ← asm.getEnvironmentState()
3:     currentS ← asm.getserviceCharacteristics()
4:     currentExperience ← (currentE, a_t, currentS)
5:     if H.containsExperience(currentExperience) then
6:         H.adjustProbability(currentExperience)
7:     else
8:         H.addNewExperience(currentExperience)
9:     end if
10:    if efficiency(g, currentS) < 0 then
11:        Invoke the decision algorithm
12:    end if
13:    sleep(Monitoring Interval)
14: end loop
```

As it can be seen in the decision algorithm, the agent history plays an important role. Indeed, this history object represents the MDP and is used in the selection of the action to be performed next. This history object is enriched throughout the life of the agent by continuous learning. The algorithm for performing the learning which is invoked periodically is provided as algorithm 2.

On the learning algorithm, it can be seen that the agent gets the current environment state from the ASM. This state characterization is performed by the ASM continuously. Additionally, the ASM also provides the learning component with a characterization of the service that is being obtained by using the service composition which is being monitored (a_t).

In order to store the informed contained in the MDP, the state of the environment, the current composition as well as the service characteristics are used together to create an experience object. This experience is then confronted with the agent's past history of experiences. The agent can already have taken into account the information contained in the current experience. In this case, only the occurrence probability for the present experience has to be modified. Otherwise, the experience must be added to the agent's history.

Finally, if the current service characteristics does not meet the applicative requirements, the learning algorithms alerts the decision algorithm that an adaptation is required.

2.3.3.6 Discussion on the extensibility of the approach

The above presented approach and algorithms do not require modification to take new or updated services into account.

One of the problems in the current networking architecture resides in the complexity that lies in the deployment of novel solutions. Indeed, new protocols are not easy to distribute. Moreover, once they are well distributed, their acceptance and usage by application developers is still not guaranteed. For example, the deployment of novel protocols such as SCTP and DCCP have been relatively slow although the work is approved and motivated by the IETF.

With the present approach, the deployment of a new mechanism is straightforward. The new service has to be included in the semiautomaton on which the system is based. For this, one solution could be to use a central repository from which end-hosts could download service descriptions as well as service implementations so they can rebuild the semiautomaton, immediately taking the new service into account.

Once the semiautomaton is updated with the required compositions, the algorithms are defined so that these new compositions are given a chance to be tested until they have been monitored at least once on every known environment. Should these new implementations lead to service characteristics which outstand the ones of existing implementations or improve the probability that the system is taken to a known good state, the decision algorithm will automatically chose the new service when faced with a decision for the same environment in the future.

2.3.3.7 Storage of experience, granularity and data structures

The above decision and learning algorithms make use of data that must be stored and updated by the learning algorithm for the decision algorithm to obtain the information it requires. In the previous sections, no assumptions were made on the size of the environment state space. In this section, the

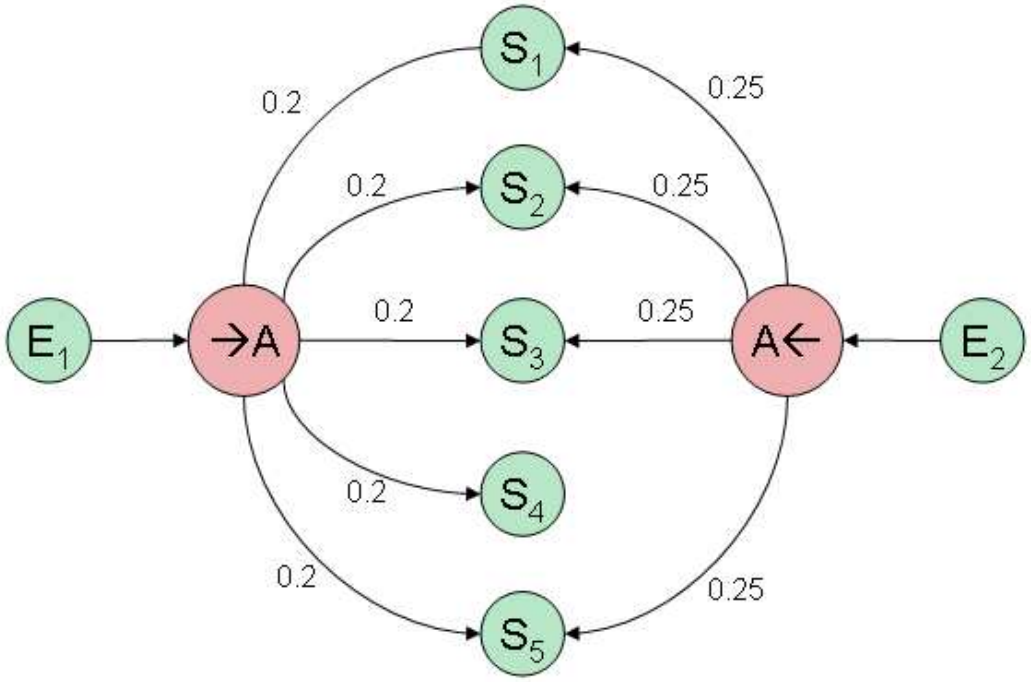


Figure 2.16: Extended MDP with coarse grained environment resolution

impact of the granularity chosen in the discretization of the state space is discussed. Additionally, an UML representation of the data structures to be used for an efficient representation and manipulation of this data is presented.

2.3.3.7.1 The importance of granularity

The environment state is characterized by three values, delay, throughput and loss rate. These values are defined and continuous over $\mathbb{R}$ which leads to an infinity of possible combinations.

In order for the learning algorithm to be efficient in finite time, it is important that the state space for the environment is as small as possible. However, for the quality of the decisions, it is important that the state definition's granularity is high enough to distinguish between two states where actions have radically opposed effects.

For example, let's consider that the environment only has two states such as the one presented on figure 2.16. E_1 defined by $T \in [0..50], D \in [0..10], L \in [0..20]$ and E_2 defined by $T \in$

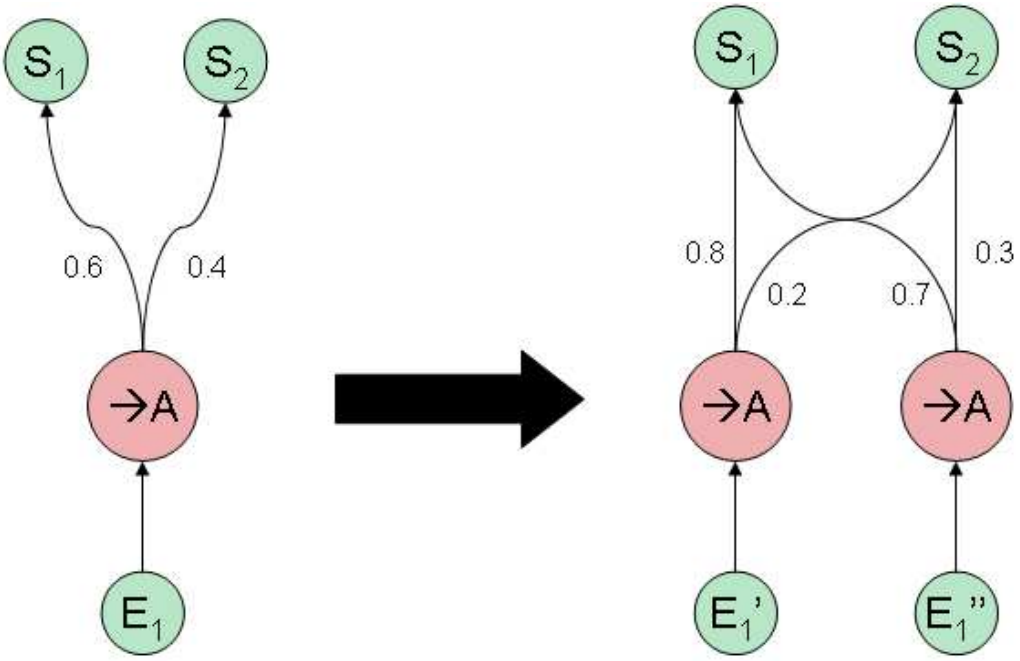


Figure 2.17: Change of environment state resolution

$[50..T_{max}], D \in [10..D_{max}], L \in [20..100]$. And let's assume that only a service providing full reliability (A) is available. In this context, the extended MDP that will be obtained from learning might be the one presented on figure 2.16. It is clear that the quality of the decisions that will be taken by examining the MDP in this context are not of much interest because of the uncertainty that there is on the resulting service characterization.

The same discussion can be lead for the granularity of the service characteristics state space. Indeed, the correct value of the discretization step is found in a trade-off between accuracy of the learned data and the speed at which this learning is to take place.

A possible solution to this problem consists in starting with a given step for the discretization process and dynamically extend or reduce the granularity depending on the context. For example, for a given value of the quantization step, if a given environment leads to two radically opposed service characteristics with similar probability such as E_1 on figure 2.16; it is likely that the present environment state needs to be subdivided. Figure 2.17 presents an example where such a solution provides a clarification of the view. Indeed, state E_1 is subdivided into E_1' and E_1'' (with $E_1 = E_1' U E_1''$) which leads to the transition probabilities to service description S_1 and S_2 to change in a

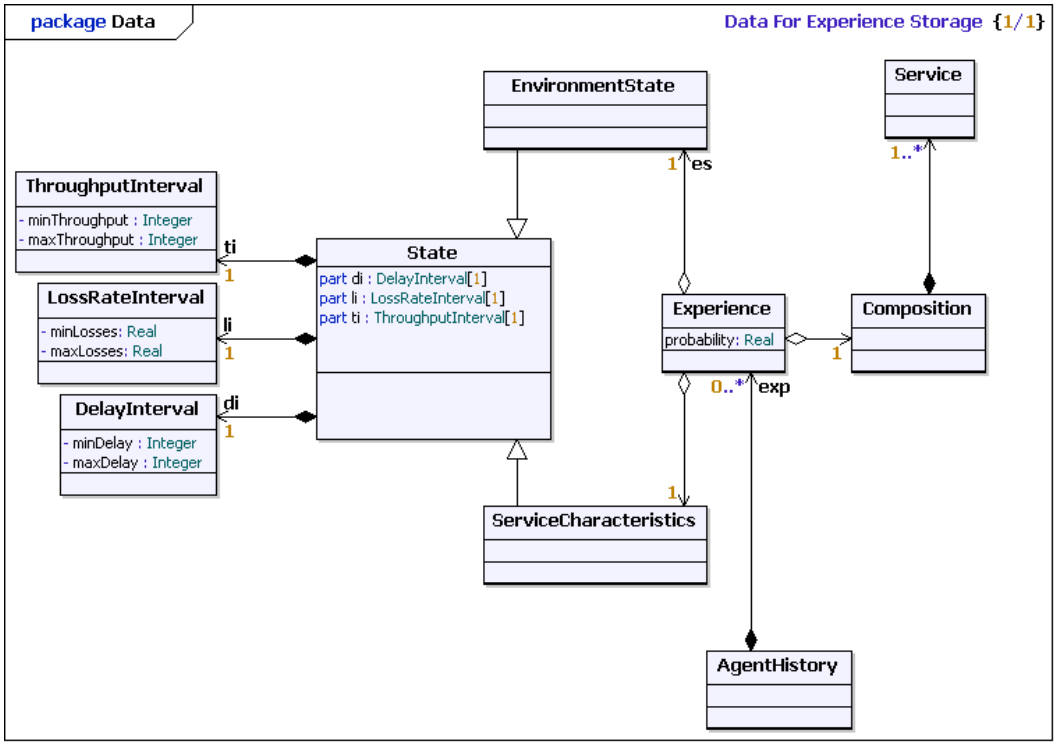


Figure 2.18: Data Structures for Experience Storage

way that makes the system more decidable.

2.3.3.7.2 Data structures for experience storage

To efficiently store the history of experiences that the agent undertakes and the result of these actions, the data structure presented on figure 2.18 is used.

On the UML class diagram, several classes take part in the process of storing the data. Indeed, to limit redundancy, *EnvironmentState* and *serviceCharacteristics* classes are provided. These classes inherit from a generic *State* class which consists in the composition of three different intervals corresponding to the values for which the environment or service is considered to be in a single state.

The *AgentHistory* models the entry point into the agent's memory. This class is a holder class composed of a set of *Experience*s. Each of these experiences corresponds to an execution of the system and is associated with a given *Composition* as well as both an *EnvironmentState* and *serviceCharacteristics*. The probability parameter of the *Experience* is used to determine the confidence

in the fact that, given an environment state and a composition, the resulting service matches the *serviceCharacteristics* defined in the *Experience* relation.

Conclusion

This chapter has presented the design of the ACT, both at the structural and the behavioral levels. After presenting the design of the architectural elements and their relation to the autonomic computing framework, the techniques and algorithms for performing autonomic learning and decision have been introduced.

The main components of the ACT have been identified at the Machine, Application and Communication Channel level. Following the *Agile Unified Process (AUP)*, their design has been specified by means of several UML diagrams. The interactions between the three levels have been identified and their design has been formulated to allow for easy extension.

The ACT architecture allows for information about the state of the different Communication Channels and Applications managed by the system to be made available, either automatically or on-demand, to identified *decision components*.

A second part of this chapter has explored candidate solutions for performing the decisions in the field of artificial intelligence. The use of *reinforcement learning* techniques to manage the service composition for a given Communication Channel has been illustrated. The algorithms for learning and taking decisions based on the gathered information have been given. The importance of the choice of the granularity of quantization for characterizing the states that the algorithms manipulate have been identified and illustrated in simple scenarios. Finally, the data structures used to store and retrieve the information acquired through learning have been described.

At this point, the ACT is able to compose services together to dynamically adapt the architecture of a protocol stack to application requirements. The composition problem for modular transport protocols presents various similarities with the one addressed in this chapter. The following chapter focuses on the implementation of the ACT by evaluating the pertinence of the models and algorithms to take the use of modular protocol frameworks at the Transport Layer into account.

CHAPTER 3

TACT: ENHANCING THE ACT TO SUPPORT MODULAR TRANSPORT PROTOCOLS

All the evolution we know proceeds from the vague to the definite...

Charles Sanders Peirce

Introduction

THE previous chapter has introduced the Autonomic Communication Toolkit. Both the models and algorithms for optimizing the service provided to applications have been discussed. The approach relies on the use of artificial intelligence and learning techniques to take experiences into account when facing situations which have already been encountered in the past. Additionally, the software architecture required to support this process and distinguish multiple levels of management (Machine, Application, Communication Channel) has been specified using UML 2.1 diagrams as well illustrated through examples.

In this chapter, the focus is made on the Transport layer. Indeed, the state of the art has introduced the possibility offered by Micro-Protocol composition frameworks to decompose the Transport layer into various sub-layers. These transport protocols can be thought of as *micro-stacks* in which every *micro-layer* implements a *service* provided by *Micro-Protocols*.

The presentation of the ACT in the previous chapter does not explicitly consider these frameworks. The existence of Micro-Protocol composition frameworks that provide custom services at a given layer of the stack is of great value to the ACT. Indeed, it allows for a finer grained adaptation of the service composition which can solve many problems without requiring a complete stack reconfiguration. This chapter studies the ACT support for such systems. Throughout the study, current limitations of the system are identified. To overcome these limitations, architectural enhancements are presented to take into account the existence of these frameworks as well as their specificities.

A first section enriches the description of the Enhanced Transport Protocol presented in the first chapter (section 1.2) and the various *Micro-Protocols* that can be composed together in this framework. Based on this description, a second section discusses the integration of these frameworks in the existing architecture and identifies the required extensions to fully support them. A third section studies the fitness of the decision and learning algorithms of the ACT to the control of such frameworks. The case of parameterized Micro-Protocols which are capable of guided behavioral adaptation is discussed in a fourth section. The stability and scalability issues that they generate when used in the ACT algorithms for decision and learning are identified and illustrated. A solution for parameter management is proposed and illustrate through an example. Finally, a conclusion summarizes the main contributions in this chapter.

3.1 The Enhanced Transport Protocol

Micro-Protocol composition frameworks have found their use at several layers of the communication stack such as Transport and Middleware [F-6]. Indeed, the composition of various software components has been used to provide elaborated functionalities from the assembly of simple building blocks.

The Enhanced Transport Protocol (ETP) [F-8] introduced in chapter 1 (section 1.2.3) follows the approach at the Transport layer. In ETP, a protocol instance is obtained by the assembly of various *Processing Modules* which cooperate together to provide adequate services to the application.

3.1.1 Definitions

In what follows, preliminary definitions required to be able to implement the algorithms for the Enhanced Transport Protocol are presented.

3.1.1.1 Processing Module

Processing Modules are passive elements in the framework. These elements all implement the same interface which contains only the *process(Message)* method. When called, this message performs a special treatment on the *Message* that is passed as parameter.

A Processing Module can for example, for each message it receives, compute and then wait for a given amount of time to shape the traffic to a certain throughput. Another Processing Module might add a sequence number to each message that goes through it.

3.1.1.2 Processing Module Container

In dynamic composition frameworks, Processing Modules are passive classes which are held in containers known as Processing Module Containers. A composition framework is generally made of several of these containers which can either be active (implemented as a thread) or passive (using another component's thread). These processing module containers are responsible for invoking the *process(Message)* method on each of the Processing Modules they contain in a well defined order when a new message is handed to them.

A processing module container can for example, be responsible for managing the workflow of processing modules' invocation from the moment a message is submitted by the application until it is sent to the lower layer.

3.1.1.3 Framework Kernel

Protocol Composition Frameworks provide a set of helper objects such as the Processing Module Containers (presented above), data storage buffers, helper objects for header field or shared variables management.

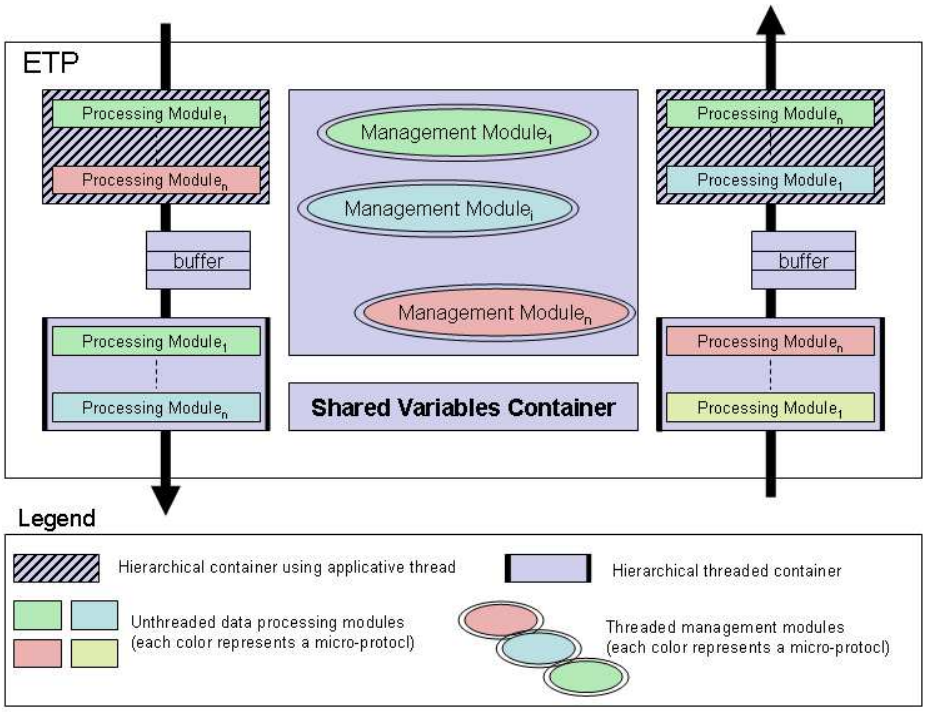


Figure 3.1: Composition architecture of the ETP

These helper objects are part of the kernel of the framework as they provide all the elements that other developers will use to implement Processing Modules that will be executed in the framework.

In ETP, the kernel contains functionalities aimed at simplifying buffer management, header construction, communication between modules using shared variables, dynamic reconfiguration as well as removal and addition of processing modules in the containers.

3.1.1.4 Micro-Protocol

Figure 3.1 presents the internal architecture of ETP. It can be seen that Processing Modules are contained in four hierarchical containers and a management container.

A given functionality (i.e., SACK-based error control, ...) can often require several processing modules to be deployed in different containers in order to provide its service. More specifically, if the Micro-Protocol needs to transmit control information from one communicating peer to another, it needs at least one processing module in the OUT container and another one in the IN container.

The set of processing modules required to provide a given functionality represents a Micro-

Protocol.

3.2 Integrating Micro-Protocol composition frameworks into the ACT

The modular composition frameworks, of which ETP is a good representative at the Transport layer, provide the user with the ability to compose different services together to tailor the resulting protocol to their specific needs. These services are materialized by the previously exposed concept of Micro-Protocol and the framework provides the necessary tools for making them work together in a coordinated fashion.

However, existing Micro-Protocol composition frameworks are not autonomic. Indeed, given a set of Micro-Protocols at their disposal, the frameworks are unable to determine which composition is best to offer adequate services to the applications and users. In most cases, the invocation of services

Definitions in Use

Jack is developing a new application and wants to use a custom made transport protocol by composing both a SACK based Error Control and a TFRC based Congestion Control.

Using ETP, Jack will provide these requirements (in XML format) to the ETP instance when he creates the socket. The ETP Kernel's dynamic loader subsystem will identify that Jack has requested two Micro-Protocol to be instantiated (SACK and TFRC). By looking among the Micro-Protocols currently available on the machine, ETP will discover that SACK is actually an Error Control and TFRC is a Congestion Control. Moreover, it will find that Error Control must be performed prior to Rate Control (by looking at dependency information provided by the developers of these Micro-Protocols). It will also find which Processing Modules implement these Micro-Protocols and the Processing Module Container they respectively have to be inserted into.

Once loaded, messages submitted by Jack's application will travel among a series of Processing Modules on their way OUT as well as on their way IN. These modules will communicate together by using Kernel functionalities in order to provide an error free, congestion friendly service.

provided by these frameworks is done by specifically describing the desired composition.

The following paragraphs discuss the use of the ACT to enhance the existing Micro-Protocol composition frameworks by including them in the architecture. The additional decision elements required as well as the coordination of these multiple decision layers in an autonomic computing framework are discussed.

3.2.1 Architectural overview

The inclusion of Micro-Protocol composition frameworks in the ACT adds a new, finer grained level of decision to the system. In a similar way that Autonomic Service Manager decisions abstract the Machine Manager from the complexity of coordinating the various services at the different levels of the stack together, the composition frameworks add a new layer that should abstract the ASM decision algorithms from dealing with the Micro-Protocols that compose these frameworks.

To support this new finer grained level of control, the ACT model presented in the first sections of chapter 2 needs to be enhanced to support this new layer of decision. This final layer of decision will be performed by the Autonomic Transport Protocol (ATP) component which will be described throughout this chapter. The following sections detail these enhancements as well as illustrate how the design provides for natural coordination between these various decision levels.

3.2.2 Adding a new decision layer to the ACT model

In the ACT model presented in chapter 2, the Autonomic Service Manager (ASM) component is described as the lower layer in the decision tree. Indeed, the ASM is described as an autonomic component responsible for coordinating non autonomic components (i.e., service instances) together.

The inclusion of dynamically configurable transport protocols in the framework adds a new level of decision to the architecture. Indeed, the ASM should now be able to deal with two kind of services, “classical” services with which it dealt before but also “autonomic services” which are able to adapt themselves to some extent. To support this, the ASM needs to be provided with the functionalities of an autonomic manager as presented on figure 3.2. In the diagram, it can be seen that the ASM now inherits from the *AutomonicManager* class. This provides the ASM with the hability to manage

instances of the *AutomonicService* class. The *AutonomicService* class present all the features of the *Service* class used in the previous chapters but also has features which belong to the *AutonomicComponent* class. An example of an Autonomic Service is the ATP which is capable of autonomically assembling Micro-Protocols together.

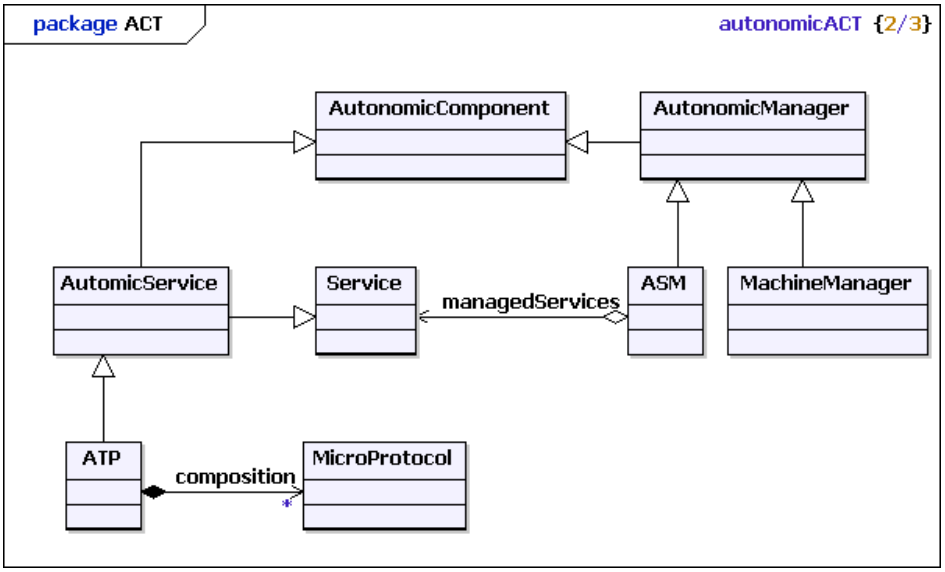


Figure 3.2: ACT architecture to support Autonomic services

3.2.3 Coordination of adaptation actions

The addition of this new decision layer to the architecture requires some precautions to be taken. For instance, caution must be taken when the ASM reconfigures the communication stack because the service characteristics are inadequate. Indeed, this reconfiguration by the ASM should not be triggered before the ATP has determined that no further adaptation is possible at the Transport layer.

The ASM algorithms for adaptation need to take into account the fact that, in case an autonomic service is present in the service composition of the global communication stack, the autonomic service's state is queried to determine whether:

- the autonomic service is aware of the performance degradation;
- the autonomic service is unable to find a suitable local composition.

Only in the case both conditions are met should the ASM adaptation algorithms be executed. Additionally, the ASM might be directly notified by an autonomic service that it is unable to provide adequate services in which case global stack adaptation should take place right away.

3.3 Decision making for the control of composition frameworks

The previous paragraphs have introduced the architecture enhancement of the ACT to support the use of composition frameworks at the Transport layer. In the enhanced architecture, these composition frameworks are considered to be autonomic in the sense that they are able to decide which Micro-Protocol composition to instantiate.

The proposed extension consists in adding a new decision layer to the ACT. This layer is materialized at the Transport layer by the Autonomic Transport Protocol (ATP). The ATP is responsible for dynamically adapting the stack of Micro-Protocols instantiated at the Transport layer to provide the service which best suits the application requirements.

The ATP implementation is based on the previously presented ETP framework for dynamic composition of Micro-Protocols at the Transport layer. To provide frameworks such as the ETP with autonomy, the following sections discuss the use of the learning and adaptation models presented in the previous section considering that Micro-Protocols are instances of services. Additionally, aspects for which no implementation guidelines have been given in the ACT presentation such as monitoring of the environment and service characteristics are explicitly described.

3.3.1 Problem identification

The present section identifies the issues in the implementation of the learning and decision algorithms presented at the ACT level to allow them to be used at the Transport level to compose Micro-Protocols together.

3.3.1.1 Build the state space

As introduced in chapter 2, section 2.3.3.1, the control entity (in this case, the ATP) needs knowledge of the semiautomaton that governs the possible executions of the agent it has to control (in this case, all possible Micro-Protocol compositions).

In Micro-Protocol frameworks, the state space consists in all the possible Micro-Protocol compositions that the framework could instantiate. However, the construction of this space is generally not trivial because dependencies (i.e., Shared Variables for communication) may exist between the Micro-Protocols. Moreover, compatibility should be taken into account to avoid the consideration of states which brake the framework's execution. This problem is further detailed in section 3.3.2.

3.3.1.2 Monitoring of service performances

Additionally to building the state space on which the controller can navigate, the decision algorithms of the ACT require that the Transport layer Micro-Protocol frameworks provide mechanisms for performing monitoring of their performance. This monitoring should be performed at two levels, just below the application and just above the underlying layer.

The measurements performed just below the application characterize the services provided by the stack as a whole (which can be compared to the application's goals). Additionally, the monitoring performed just above the underlying layer characterize the state of the environment over which the transport instance is to be deployed.

Considering the information required by the decision processes in the ACT, both measurements need to be taken into account. On the one hand, they allow to characterize the environment on top of which the agent should instantiate its compositions. On the other hand; they evaluate the adequation of the controller's decisions with regards to the application goals and thus can be easily used to compute the reinforcement. Section 3.3.3 presents a detailed discussion on monitoring for adaptation and learning.

3.3.2 Composition model

To be able to make control decisions, the ACT needs to be provided with a semiautomaton that describes the possible states of the agent it controls. In the present case, the various states are materialized by all the *possible* Micro-Protocols compositions given a pool of Micro-Protocols and their respective dependencies.

In the context of self-adaptiveness for dynamically configurable communication protocols, this thesis introduces a modeling of the decision process for building the state space from a given set of modules to be instantiated in compositions. This modeling allows for automating the decision for architectural adaptation of communication protocols. This process can then be referred to as *run-time* self-adaptation.

The composition model is aimed at defining the conditions of the validity of the assembled Micro-Protocols. The conditions for this validity will be used to reduce the size of the semiautomaton in which each state represents a potential composition candidate for the decision algorithms.

As presented earlier, the building block of any dynamically configurable communication protocol is the Micro-Protocol. By definition, a building block can only realize one elementary function. A protocol is obtained by the successive invocation of Micro-Protocol each providing a specific service in a predefined order. This constitutes the main hypothesis of the composition model.

A communication protocol can be divided into L functional levels. When it comes to composition, for each level (i), there is a set (D_i) composed of L_i protocol modules each providing a different implementation of the same elementary service.

Definition: Let $D = \{\cup_i D_i\}_{i=1..L}$ with $D_i = \{A_j\}_{j=1..L_i}$ representing the set of protocol modules providing a service specific to level i.
D is the universe of protocol modules available for composition.

Definition: Let A, a protocol module. To be able to perform correctly, A requires information published by other protocol modules (i.e., sending rate, packet loss rate, ...) and itself publishes information that might be of interest to others. This is represented by $A = [A^P A^R]$, where:
A^P, the publication vector of A such that:

$$A_i^P = \begin{cases} 1 & \text{if } A \text{ publishes variable } i \\ 0 & \text{otherwise} \end{cases}$$

A^R, the requirements vector of A such that:

$$A_i^R = \begin{cases} 1 & \text{if } A \text{ requires variable } i \\ 0 & \text{otherwise} \end{cases}$$

Let $C = \{(A)_1, \ldots, (A)_i, \ldots, (A)_N\}$ a composition of N modules issued from D sharing information via M shared variables. In what follows, we use the previously introduced structures to define the validity conditions for module compositions. This instantiation of the composition model is given only as an illustration but can be further refined to any particular purpose as explained hereafter.

$$C\text{-}valid \Leftrightarrow \begin{cases} (\sum_i (A^P)_i)_k \leq 1 & \forall k \in [1..N] \\ \sum_i (A_k^P)_i > 0 & \forall k : \sum_i (A_k^R)_i > 0 \\ \forall i : A_i \in D_i \text{ then} & \nexists A_j \in C : (A_j \in D_i) \end{cases} \tag{3.1}$$

The first condition allows to guarantee the absence of conflicts in the publication of shared values in the sense that it does not allow two modules of a composition to publish the same variable. This allows for guaranteeing the coherence of the publication for scenarios where protocol module conceivers did not take concurrent publication issues into account. The second condition allows to verify that all variables required by at least one module are published by another one. This guarantees that all modules instantiated in C have all the information they require available for them at *run-time*. The last condition guarantees that, at most, only one module is instantiated at each of the identified levels. The composition model can be further extended to take other elements into account (i.e., interdependency of modules, compulsory inclusion of specific modules such as those performing monitoring in all compositions, ...). This extension is achievable by the addition of new constraints to the above system.

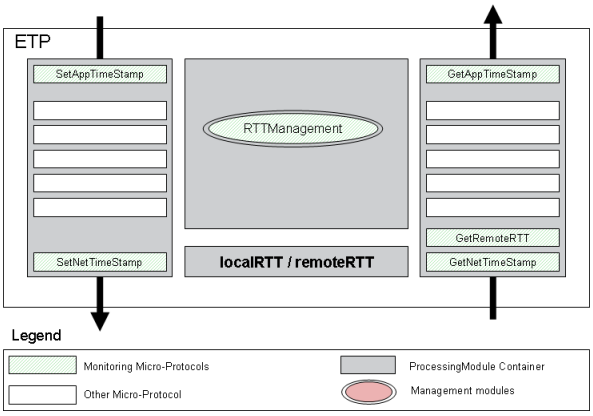


Figure 3.3: RTT monitoring in ETP

3.3.3 Monitoring for adaptation and learning

The algorithms for adaptation and learning presented in chapter 2 remain valid when dealing with dynamically configurable transport services. Indeed, the two situations are instances of the same problem but at different levels of detail. The problem addressed in chapter 2 concerns service composition for the complete communication stack by considering that a service is provided either by a protocol instance or a QoS framework (i.e., EuQoS or NetQoS ...). In this chapter, the services are provided by Micro-Protocols which can be composed together to form a protocol instance.

To be able to maintain the best possible service for the application's communication channel, the performance of the actual service composition in the ETP as well as the state of the environment on which it is being executed have to be monitored. The monitoring can take advantage of the modular architecture of ETP to be easily deployed. Indeed, the composition model presented can be adapted so that all compositions included in the semiautomaton it generates contain selected monitoring Micro-Protocols.

3.3.3.1 Measuring applicative delay and network RTT

The monitoring of the network RTT as well as the applicative delay can be easily performed by using the application generated data packets and simply adding specific header information to them. Figure 3.3 presents the different building blocks that compose the network RTT and applicative delay monitoring Micro-Protocol.

As it can be seen on figure 3.3, several processing modules are required to monitor these values.

3.3.3.1.1 Network RTT

To monitor the network RTT, the ETP instance executing on the sender side adds a header field that contains a timestamp indicating the date at which this packet was delivered to the lower layer (t_{sent}). When this packet is received, the ETP entity records the date $t_{received}$ and associates it with the t_{sent} value it retrieves from the packet's header fields.

As clocks of Internet hosts may not always be synchronized, the RTT value needs to be computed by the same entity that set the t_{sent} timestamp. To achieve this, the receiving entity generates feedback data (possibly by piggybacking) in which it sends the t_{sent} timestamp as well as the duration between $t_{received}$ and the moment the feedback packet is delivered to the lower layer ($t_{process}$). The RTT can then be computed upon reception the sender host as : $RTT_{local} = (now - t_{sent}) - t_{process}$. The computed RTT_{local} value is then shared with the peer entity for which it corresponds to RTT_{remote}.

3.3.3.1.2 Applicative delay

Once the network RTT is known, it is relatively easy to obtain an approximation of the delay required for application messages to be delivered.

Indeed, each time the sending application submits data to the ETP instance, the date is recorded as a header field ($t_{AppSubmit}$). Once the applicative message is delivered to the receiving application (possibly after several retransmissions), a feedback packet is generated by the receiver (by piggybacking when possible). This feedback packet contains the ($t_{AppSubmit}$) value. Once this feedback is received, the sending entity is able to approximate the applicative delay as: $D_{local} = (now - t_{AppSubmit}) - \frac{RTT_{local}}{2}$.

However, it is important to note that this only constitutes an approximation of the actual applicative delay. The accuracy of the estimate improves with the symmetry of the delays on the forward and return paths between the entities.

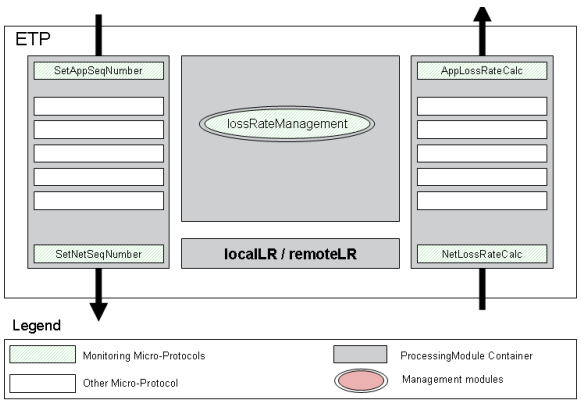


Figure 3.4: Loss Rate monitoring in ETP

3.3.3.2 Measuring application and network loss rate

In addition to the monitoring of the delays and RTT, the environment and service's packet loss rate also characterize these two elements. The monitoring of these values is relatively simpler than the RTT. Figure 3.4 presents the different building blocks that compose the network and applicative loss rate monitoring Micro-Protocol.

3.3.3.2.1 Network loss rate

To monitor the network loss rate, a Micro-Protocol is instantiated in the lower parts of the ETP architecture. This Micro-Protocol adds a sequence number to every packet that is transmitted by the protocol entity. A packet resulting from a retransmission will receive a new sequence number.

On the receiving side, a second processing module for the Micro-Protocol is instantiated. It maintains a view of the sequence number for the packets received during a given period of time. Using this view, it is able to compute a sliding average of the packet loss rate. This corresponds to the local packet loss rate. This value is transmitted to the peer entity for which it is saved as the remote loss rate.

3.3.3.2.2 Applicative loss rate

The monitoring of applicative loss rate is similar to the monitoring of network loss rate. However, the packet numbering as well as the packet loss rate computation modules are placed just below the

application. This allows for packets that suffer from one or several retransmissions to keep the same sequence number as they are not processed by the numbering processing module more than once.

3.4 Considering parameterized Micro-Protocols

In ETP, as well as other Micro-Protocol compositions frameworks at the Transport level, Micro-Protocols can be of two kinds. The most common type of Micro-Protocol is the one considered until now which provides basic services such as error control (i.e., *full reliability using SACK vectors*) or congestion control (i.e., *TFRC*). However, some Micro-Protocols provide more elaborated services which take a parameter into account. For example, the Micro-Protocol performing *partial reliability using SACK vectors* requires a parameters that sets the required reliability level.

This section discusses various approaches to take the parameterized Micro-Protocols into account. The analysis of these solutions performed hereafter reveals that some approaches might have negative effects on the stability and scalability of the learning and decision algorithms.

3.4.1 The multiple services approach

In order to consider Micro-Protocol parameters values in compositions, several approaches exist. Indeed, the simplest approach could be to consider each <Micro-Protocol, Parameter Value> pair as a new service which would then be used to build the state space for the possible compositions.

For example, for the partial reliability Micro-Protocol, if the reliability parameter can be considered discrete over its range of possible values $[0..1]$ by steps of 0.1, the result of applying this method is to consider 11 additional Micro-Protocols in the pool of available Micro-Protocols. If we consider a pool of Micro-Protocols of size 2 prior to doing this, with no compositional restrictions, the number of states increases from 1 (TFRC + SACK) to 11 (TFRC+SACK[0], TFRC+SACK[0.1]..., TFRC+SACK[1.0]).

With the previously presented approach, the increase of the state space size that is generated by the discretization of the Micro-Protocols parameters might lead to performance issues such as slower learning, oscillations and situations where no solution is available.

While the reason for the learning to be considerably slower can easily be pictured as a direct consequence of the increase in the state space size, oscillations are harder to capture. The following example illustrates their existence.

Let there be a universe of modules (D) composed of a single module M. M is a parameterized module which requires a value t to be passed upon instantiation ($t \in \mathbb{R} \cap]0..0.2]$). The discretization of t by steps of 0.1 leads to considering a universe (D') composed of 2 modules leading to 2 possible compositions each containing a single module but having a different parameter value. In this context, two possible MDP instances obtained through learning are represented on figure 3.5.

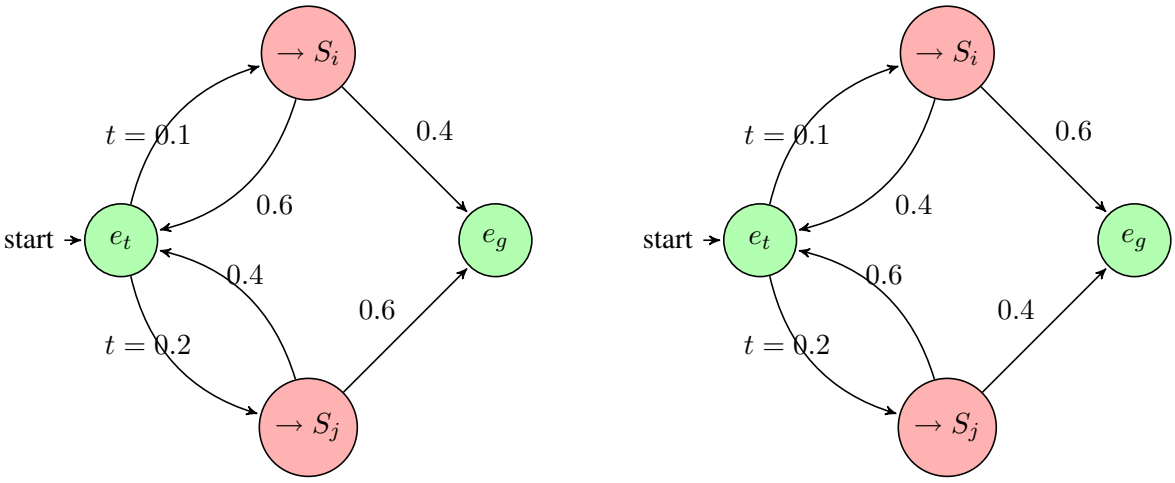


Figure 3.5: Discretization osciallations

In this example, oscillations appear as a result from the discretization that was performed. Let us suppose that the ideal value for parameter t is 0.15, the discretization has lead to only two values being considered. To illustrate the problem, the environment is considered to have only two states, e_t which is the current state and e_g which is the state that fulfills the goals. Let us suppose that the system has the current MDP representation presented on the left of the figure. The agent decides to apply composition S_j which maximizes the probability to obtain the environment state e_g. Because the value of 0.2 for parameter t is not optimal, results are uncertain and do not always lead to e_g. In this case, the learning algorithm might lower $P(e_g|S_j)$ and increase $P(e_t|S_j)$, the next time the agent is confronted with the same decision, it will prefer to chose composition S_i. Once more, as

the parameter value of 0.1 is not optimal, the result might not be as expected and cause the MDP to oscillate between the two forms presented on the figure.

This situation is mainly due to the fact that the MDP does not contain compositions in which the correct parameter value is present. Indeed, if a parameter value of 0.15 takes the system into state e_g with a probability of 0.9 when its current state is e_t, the oscillation problem would not exist.

Another possibility is that no composition improves the situation. Because the discretization reduces the possible parameters values, an MDP such as the one presented on figure 3.6 might be learned by the agent. In this case, the absence of a composition in which the adequate parameter value is considered leads to situations where no solution is available. Indeed, it can be seen that any action performed in state e_t has a high probability of being ineffective in the sense that the environment will remain in state e_t.

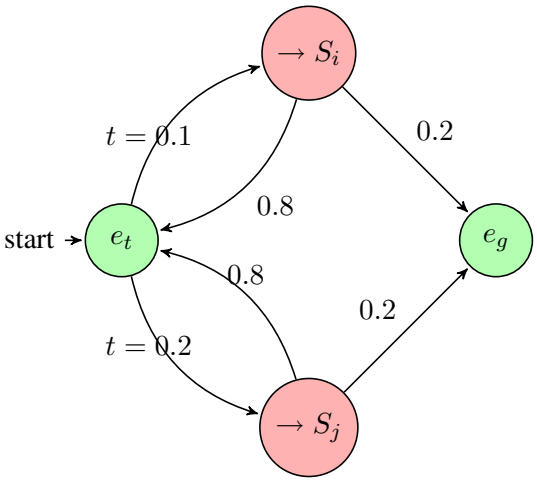


Figure 3.6: Discretization effects: no suitable parameter value

To support the dynamic composition of Micro-Protocols which provide a service which is dependant of the value given to a parameter during their instantiation, the intuitive solution that consists in treating each possible parameter value as a new Micro-Protocols has been shown to be the inadequate. Indeed, the increase in the state space induced by this method does not scale well and can lead to sub-optimal performances such as longer learning time. Moreover, the discretization of the parameter value required to limit the increase of the state space has been shown to be a source for

oscillations as well as solutionless situations.

In the following paragraphs, another approach is followed to take these Micro-Protocols into account. This approach is based on the conclusions drawn from the above presented limitations and constitutes a logical extension to the ACT framework.

3.4.2 The autonomic Micro-Protocol approach

The following sections detail an approach relying on the autonomic computing paradigm that recommends delegation to reduce the complexity of the decision process by keeping the high level generic and specializing on the way down. The contribution hereafter introduces autonomic Micro-Protocols which constitute an ATP enhancement to ETP. The Autonomic Transport Protocol (ATP) which is based on the ETP kernel but is also able to load and manage autonomic Micro-Protocols.

The ETP structure is used as a base for the development and instantiation of autonomic Micro-Protocols in ATP. For this, the ATP extends ETP by providing several monitoring Micro-Protocols in its core distribution (which allow for measuring various network as well as applicative metrics). These monitoring Micro-Protocols are a key value to the development of Autonomic Manager components located in the management plan.

The autonomic manager modules are built following control theory for complex systems. Indeed, the Autonomic Manager is seen as a controller which uses data collected by the sensors (monitoring micro protocols) to determine the correct value of the command (micro protocol parameters) to be applied to the actuators (processing modules) to satisfy the applicative goals. This view of the ATP is particularly well suited for its inclusion in the ACT to which it adds a new level of decision.

3.4.2.1 Autonomic control model

Several methods exist for controlling complex systems. However, most of the existing methods require that a model of the controlled system exists in order to be able to best predict the system response to a given command. In networking however, obtaining a model of the system is a complex task as many factors influence the system response to a given stimuli. Exact models can therefore only be obtained in restricted environments for which only a limited set of parameters are taken into account.

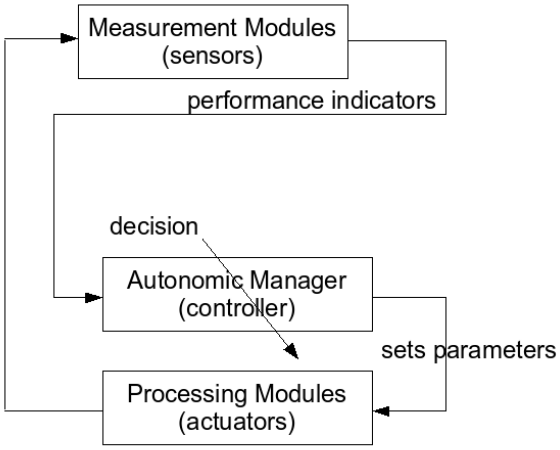


Figure 3.7: ATP Control System: Autonomic Manager

Once the environment differs from what was predicted “a priori”, the system no longer performs as expected.

When the complete model of the system cannot be established, methods such as the intelligent control which rely on general trends and global behavior estimation provide ways for adapting the command based on the current system’s performances.

The control model used for the Autonomic Manager is presented on figure 3.7. The Autonomic Manager is the central entity in the feedback control loop. Indeed, the Autonomic Manager is responsible for tuning the parameter values of the Processing Modules (which act as the actuators of the autonomic component). To perform this task, it relies on the inputs provided by both the measurement modules (which act as the sensors of the autonomic component) and the learning module.

3.4.2.2 Architectural overview of an autonomic Micro-Protocol in ATP

This section discusses the ATP architecture to integrate Micro-Protocols which are able to automatically adjust their parameter value. These Micro-Protocols follow the conceptual model presented on figure 3.7.

The autonomic manager has two main tasks: define the values of the parameters of the different ProcessingModules that implement a given Micro-Protocol and determine the adequation between

the service the Micro-Protocol is providing and the goals set by the ACT.

To set the parameter values for the different ProcessingModules, the Micro-Protocol designer can use the Shared Variables framework provided by the ETP kernel to externalize these configuration values.

The adequacy between the service being provided and the goals set by the ACT is harder to implement. Indeed, the goals are generally expressed as a set of QoS parameters as defined during the ACT design. However, the ETP framework does not provide monitoring Micro-Protocols natively. The ATP natively bundles and supports these monitoring Micro-Protocols. Depending on which level of the stack they are instantiated at, the values they measure will reflect either application or network level metrics. Indeed, while only application level metrics are useful to evaluate the service's efficiency, network level metrics are useful to guide the decision process by characterizing the environment.

For example, an autonomic manager modifying the loss threshold of a partial reliability Micro-Protocol to minimize applicative losses while avoiding important increases in the applicative transit delay might use the network level observed loss rate to compute the average number of retransmissions that will be required and thus estimate the resulting applicative delay.

Figure 3.8 presents the integration of autonomic Micro-Protocols in the ATP architecture. The modules instantiated in the data plans (IN and OUT) are identical to their non-autonomic counter-

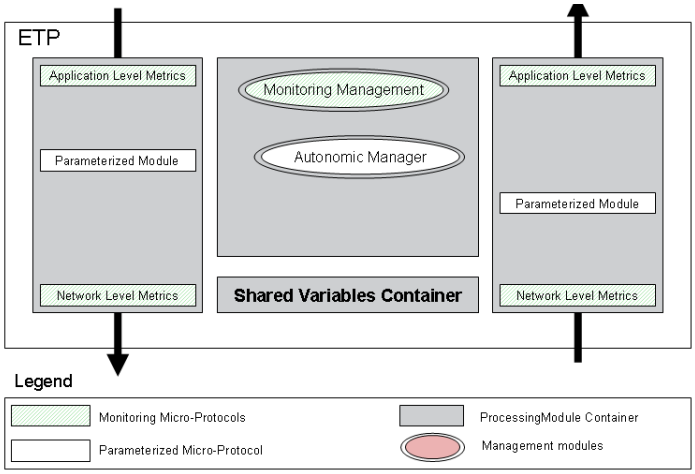


Figure 3.8: Autonomic Micro-Protocols in the ATP Architecture

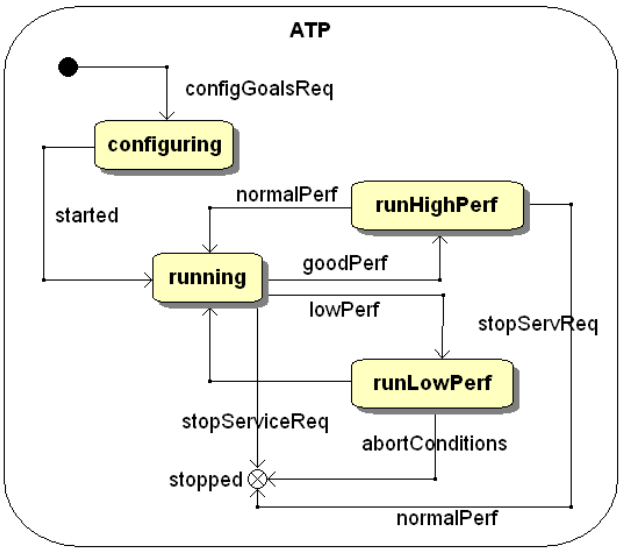


Figure 3.9: ATP State Machine

parts. The addition of the Autonomic Manager is made in the management plan as an active element. Even in the non-autonomic scenario, parameters are passed to the Micro-Protocols by using shared variables, the Autonomic Manager only has to modify the value of these parameters to alter the Micro-Protocol's behavior. Additionally, monitoring Micro-Protocols can be seen both at the top (just after the application submits its ADU to ATP) and the bottom (just before the PDU is handled to the lower level) of the stack. This allows both application and network level metrics to be gathered.

3.4.2.3 Autonomic management module automata

The autonomic management module is the central element of autonomic Micro-Protocols. To achieve its goal, its implementation is located in the management plan of the ETP Kernel. This allows it to monitor and adapt the services offered by the ATP managed Micro-Protocols. This location gives the manager access to all the kernel functionalities for issuing commands to the different Micro-Protocols that are instantiated in the ATP. For example, the manager can decrease the tolerated loss rate of an ARQ (Automatic Repeat reQuest) mechanism or query the status of a measurement module performing network packet loss rate estimation.

The behavior of the manager is described by the automaton presented on figure 3.9. Initially,

when the ACT instantiates autonomic Micro-Protocols, it passes QoS goals for the connection. These QoS goals are expressed as a set of QoS parameters (applicative losses, applicative delay, average throughput...) to which one or more values are attached. Additionally, the ACT can prioritize its requirements to give information on which requirement to satisfy first when it is not possible to satisfy them all.

The first state of the manager upon creation is the “configuring” step in which the Micro-Protocols are instantiated and initial parameter values are set. Once the application has started sending data, the manager is initially in the “running” state. In this state, the Autonomic Manager will use information provided by the measurement modules (sensors) to compute and evaluate performance indicators which will be compared to the optimal ones to determine the quality of the current command. Depending on the output of the evaluation, the module state will be shifted to “runHighPerf”, “runLowPerf” or maintained at “running”. The ATP Manager will stay in the “running” states while the monitored transport service remains in the target goals. The “runHighPerf” and “runLowPerf” states will be reached if the monitored transport service performance exceeds high or low level performance thresholds.

Adaptation strategies aimed at modifying the transport composition should be performed in the "runLowPerf" state in order to improve the service. For this, the manager will trigger the sending of messages to the *ASM* indicating that it is not fulfilling the goals that have been set. If the manager remains in “runLowPerf” for too long, the *ASM* might trigger a change in the service composition. Notifications about the good performance are sent to the *ASM* while the manager stays in the “runHighPerf”. These are used to adjust the history component of the learning model. Additionally, this information can be used to take higher level decisions such as multi-flow or multi-application optimisations. The normal "running" state can be reached again, if the service performance gets back to normal targeted values.

3.4.3 Example parameter management algorithm

In this section, an algorithm for managing the reliability parameter of an error control mechanism is provided as an illustration of the concept. The goal is to maximize partial reliability while maintaining

Algorithm 3 Parameter Control Algorithm

Require:

The target delay D_t.
The current delay D.
The delay on the previous algorithm invocation D_{last}.
The maximum loss rate L_{max}.
The ARQ mechanism partial reliability parameter L_{ARQ}.
The current loss rate L.

1: $D_{kpi} \leftarrow \frac{D - D_{last}}{TIME\_STEP}$
2: $L_{kpi} \leftarrow \frac{L_{max}}{L}$
3: **if** $D_{kpi} < 0$ **then**
4: The current delay value allows for more retransmissions
5: **if** $L - LR\_STEP > 0$ **then**
6: **if** $|D_{kpi}| > D\_CONFORT$ **then**
7: $L_{ARQ} \leftarrow L_{ARQ} - K * LR\_STEP$
8: $agentState \leftarrow runHighPerf$
9: **else**
10: $L_{ARQ} \leftarrow L_{ARQ} - LR\_STEP$
11: **end if**
12: **end if**
13: **else**
14: The current delay value requires that more losses be tolerated
15: **if** $L + LR\_STEP <= L_{max}$ **then**
16: **if** $|D_{kpi}| < D\_CONFORT$ **then**
17: $L_{ARQ} \leftarrow L_{ARQ} + K * LR\_STEP$
18: $agentState \leftarrow normal$
19: **else**
20: $L_{ARQ} \leftarrow L_{ARQ} + LR\_STEP$
21: **end if**
22: **if** $D > D_t$ **then**
23: $L_{ARQ} \leftarrow L_{max}$
24: $agentState \leftarrow runLowPerf$
25: **end if**
26: **else**
27: $agentState \leftarrow runLowPerf$
28: **end if**
29: **end if**

the delay below a given limit. This is based on the assumption that having a delay 10 times smaller than the announced goal while experiencing loss rates close to the announced limit is not as good as having a delay closer to the high limit while maintaining control over losses. This is the case for multimedia applications which perform a small buffering of the stream.

Algorithm 3 presents a pseudo-code transcription of such mechanism. As it can be seen, the mechanism follows a straightforward approach inspired by the PID (proportional-integral-derivative) controller from the theory of control. The goal is to derive a command that will be applied to the partial reliability mechanism as its parameter value. To do so, the applicative delay variation is observed (derivative part of the model). Based on the observed trends in the delay variation, the command is adapted proportionally to the distance between the sensed values and the target (integral part). This algorithm has been implemented as a proof of concept in a prototype of the ATP, the results obtained for this mechanism are presented in section 4.3 of the next chapter.

Conclusion

In this chapter, the architecture of the ACT has been extended to provide support for services which are capable of a certain level of self-adaptation. This support is achieved by extending the ASM for it to provide autonomic manager functions. Following this approach, the Autonomic Transport Protocol (ATP) framework has been defined around the ETP kernel to provide it with autonomic component functionalities. Indeed, the ATP is able to dynamically perform architectural adaptation of the Micro-Protocol stack while also fully supporting Autonomic Micro-Protocols which are capable of a certain level of behavioral adaptation.

The ACT definition given in chapter 2 considers adaptation to be limited to its architectural form. The study performed in this chapter illustrates the need to support both architectural and behavioral adaptation to take advantage of the full extent of possibilities offered by the services being composed.

For architectural adaptation, the learning and adaptation algorithms presented for the ACT have shown to be suitable for use at the Transport layer without being modified. The mechanisms for characterizing both the environment as well as the service resulting from a given Micro-Protocol composition have been clarified in the context of a Transport protocol. The interactions between the ASM and the newly introduced ATP have been described to avoid the conflicts that arise when two different decision elements address a given problem. The approach consists in always trying to resolve situations as low as possible in the architecture (namely, the ATP changing the Micro-Protocol

composition) before a higher level solution is sought (at the ASM level where the complete Stack is modified).

Behavioral adaptation consists in changing the value of a given Micro-Protocol parameter to modify the service that it provides. For behavioral adaptation, the use of MDP based approaches has shown to be unadapted, causing oscillations and scalability issues. To overcome this limitation, the introduction of autonomic Micro-Protocol has been proposed. These Micro-Protocols include a manager that will modify the values of command parameters when needed based on either a theoretical or empirical model. An example of a control algorithm following the proportional-integrative-derivative automated control model has been illustrated.

In the next chapter, the architecture and algorithms for behavioral and architectural adaptation are evaluated. This evaluation is performed both in simulated and emulated environments to illustrate the concepts and mechanisms presented previously.

CHAPTER 4

EVALUATION

A theory is something nobody believes except the person who made it. An experiment is something everybody believes, except the person who made it...

Albert Einstein

Introduction

THE previous chapters have introduced the ACT, a toolkit containing both architectural principles and implementation guidelines for coordinating services provided at various levels of the communication stack to optimize the user perceived quality of service. Additionally, the integration of dynamically composable transport services into the ACT through the addition of a new decision level materialized at the Transport layer by the Autonomic Transport Protocol (ATP) has been discussed.

In this chapter, several aspects of the ACT, its architecture as well as the models that drive its operation are evaluated at the Transport layer. Whenever possible, this evaluation is performed in a real network environment tested. However, for a proof of concept or when some of the aspects cannot be evaluated in a real network, simulation is used. When used in a real environment, the prototype of the ACT that is evaluated is capable of composing services at the Transport layer by using the Autonomic Transport Protocol.

The next sections are organized as follows. After presenting the test platform which will be used to perform the real network evaluations as well as the tools that are available for such testing,

the performance and benefits of the ACT are evaluated. A first section presents a proof of concept which validates the learning approach by illustrating the construction of an MDP through simulation. Following the proof of concept, the suitability of the Autonomic Micro-Protocol solution to perform behavioral adaptation is shown in simple scenarios where traditional services are unable to achieve acceptable results. The construction of knowledge for architectural adaptation by the ACT is illustrated. The use of this knowledge by the decision algorithms is studied and explained for a test scenario in which the environment parameters are highly variable, stressing the system. Finally, a study of the overhead introduced by the ACT and a discussion on scalability are presented.

4.1 Test platform and tools

The evaluation presented hereafter has been performed on a network platform that aims at emulating varying network conditions. The following sections describe the architecture of the platform as well as the tools used for automating the deployment, measurements as well as result presentation.

4.1.1 Test platform presentation

The test platform used for the evaluation of the ACT is presented on figure 4.1. It is implemented as part of the LAASNetExp initiative for network experimentations. All the machines on the platform have the same hardware configuration (Dual Xeon 3050 at 2.13Ghz, 2GB ECC 667Mhz RAM) and all physical interfaces operate at 1Gbps (2*Broadcom 5721J and 2*IntelPro1000PT). The network uses Gigabit Ethernet and is implemented using Cisco Catalyst 6504 as well as 2960 switches. VLANs are used to isolate the different networks. The switches are trunked using 10Gbps links but no VLAN spans across switches except for the management network. The machines are installed with Ubuntu Linux 7.10 server running Linux kernel 2.6.22.

The platform is composed of four distinctive IP networks. Three of these networks are used for experimentation while the fourth one is exclusively reserved for management traffic (ssh sessions, NTP synchronization, routing, ...). The *pong3* machine acts as a router among the different experimentation networks while *pong1* is the gateway to the Internet (through which the platform is

accessed).

To re-create various network scenarios, emulation is used. For this purpose, all experiment traffic should be routed through *pong3* on which the Linux Network Emulator (NetEm) [M-14] is installed. NetEm allows an experiment to artificially delay, space, randomly drop, shape or police the streams that flow through it.

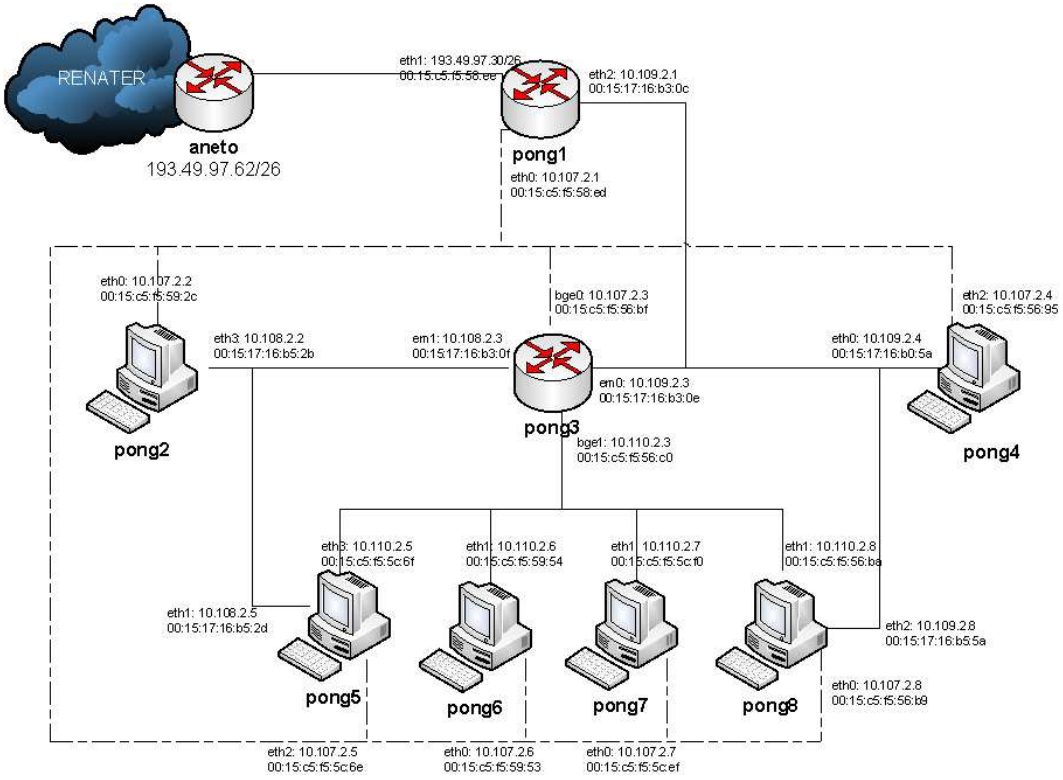


Figure 4.1: Test Platform Architecture

4.1.2 Tools for automated protocol testing

Evaluating the behavior of network protocols is generally a difficult task. Indeed, previous experience has shown that thorough testing is required in order to verify that the solution being proposed performs as announced in all the environments that are said to be supported.

For example, the design of the TCP congestion and flow control mechanisms originally targeted networks where the $Delay \times Bandwidth$ product was low. In such networks, the convergence time of these algorithms was reasonable. However, further tests in networks with a higher $Delay \times$

$Bandwidth$ product have shown disastrous performance in regards to time of convergence to steady state.

To improve the quality of the designed protocols and communication systems, the following sections present both simulation tools that can be used to improve the quality of the designed solutions as well as the MINER platform for monitoring and measurements.

4.1.2.1 Model Driven Architecture and model simulation

In protocol design, several approaches exist to provide design time evaluation of the performances of a given architecture. Formally proven languages and methods such as CCS, LOTOS and Petri Nets [M-21] provide tools that allow for architecture and behaviors to be specified and verified using Linear Temporal Logic (LTL) queries [M-1]. Indeed, given a model, several basic properties can be formally proved.

Although very useful to validate the concepts and algorithms used for designing protocols, formal techniques have two main drawbacks. The main limitation is related to the complexity of the system that can be modeled. Indeed, combinatorial explosion and infinite state space are common obstacles to the verification of complex protocols such as TCP. On the other hand, in order to avoid these problems, models are generally simplified. This leads to a bias to be introduced between the actual implementation and the model on which the verifications have been performed providing no guarantees on the final implementation.

With the advent of software modelling techniques around UML, new methods for managing the workflow such as Model Driven Architecture (MDA) have been introduced. With MDA, the design phase is highly detailed and each component's behavior is described by the use of state machines. These state machines can then be used by specific tools to generate an executable artifact. This artifact can then be interacted with to simulate the behavior of the component being tested. Although weaker than the output of a verification tool, the outcome of extensive simulations can be used to detect misbehavior and design flaws at an early stage of development.

The TAU Generation 2 tool edited by Telelogic provides an environment for MDA. TAU includes both model editors for all the UML diagrams available in the 2.1 version of the UML norm as well as

an interactive model simulator. The model simulator uses the state machine diagrams which describe the behavior in conjunction with the composite structure diagrams which describe the structure in order to generate an artifact. This artifact can then be interacted with by sending signals which contain commands. All the internal execution steps can be traced by using an interface similar to the one of a debugger which navigates through the different models as the system is executed.

For example, using TAU, an ETP Micro-Protocol composition can be specified in UML. This component can then be interconnected with other components also designed in UML which simulate an application sending data on the one side and a network medium on the other side. By generating an artifact of such system, the designer can verify that the model behaves as expected in various network environments and potentially correct misconceptions prior to any implementation.

This method is used in the following section to evaluate some of the aspects of the ACT architecture.

4.1.2.2 MINER: automated protocol testing

MINER [M-4] is a distributed monitoring and measurements architecture developed by Salzburg Research. Its primary goal is to establish an infrastructure that provides high-quality support for research. The declared goal of the MINER framework is to provide a higher-level infrastructure that is able to integrate existing monitoring tools as well as extensible to allow new tools to be integrated.

The MINER architecture relies on two main components, the MINER Core and one or several MINER Tool Proxies. The core is the centralized entity responsible for coordinating the various measurement scenarios that take place on the testbed. To do this, the core relies on various proxies which are executed on the measurement points. The Tool Proxies are containers which instantiate the various measurement tools and collect the results provided by these tools to forward them to the core for storage.

To execute a measurement, a scenario has to be submitted to the core. This scenario (XML file) will contain a description of the architecture on which the measurements will take place, indicating the proxies involved, the tools to be deployed on each of these proxies as well as the parameters that need to be passed to these tools upon instantiation. The core will then take all the necessary steps for

the scenario to be accomplished and the results to be stored.

In the testbed, all MINER related signalling which takes place between the core and the proxies is performed on the management network for this traffic not to interfere with the experimentation being performed.

In order to monitor the performance of the ACT and ETP on the testbed, several specific tools have been developed for MINER. A first tool has been developed to launch ETP and ACT instances on a specific machine. A second tool has been created to query the ETP and the ACT for the values of specific shared variables which will be forwarded to the core for storage. In order to perform this monitoring, specific modules for the ETP and the ACT have been implemented. These modules interact with the MINER tool to answer queries they receive for given variable values.

Finally, as protocols need to be tested on varying network conditions, a specific tool which instantiates the Linux NetEm framework with a set of given parameters has been used on the *pong3* machine. A simple utility for specifying and generating a set of scenarios, and submitting them to the core for execution has been elaborated. This utility, which takes an XML file as input, allows for example to specify that a given ETP composition is to be tested in all combination of delays and losses with delays varying between D_{min} and D_{max} in increments of D_{inc}, and losses varying between L_{min} and L_{max} in increments of L_{inc}. This will lead to the composition being tested with NetEm parameters set as $\{[D_{min}, L_{min}], [D_{min} + kD_{inc}, L_{min}], \ldots, [D_{min}, L_{min} + kL_{inc}], \ldots, [D_{max}, L_{max}]\}$.

4.2 Proof of concept through simulation

In this section, the concepts presented in the previous chapters are put into practice through simulation using the above presented platform and tools. This first study is intended to demonstrate the feasibility of the approach, functional validation of the decision and learning algorithms as well as completeness of the design. After instantiating the composition model presented in section 3.3.2 on a simple example, various scenarios performed in simulation are analysed and evaluated.

4.2.1 Building the state space and semiautomaton

To be able to decide which composition to instantiate, the Autonomic service Manager (ASM) of the ACT needs to be provided with a semiautomaton that describes all the possible service compositions. The following paragraphs demonstrate the construction of the semiautomaton in the context of dynamically configurable transport protocols. The model presented in section 3.3.2 is used to create the semiautomaton used for the scenarios evaluated in simulation.

For illustration purposes, we consider a multimedia transmission. For this example, the transport protocol's internal architecture is divided in three simple functions (i.e., layers of protocol modules). First, the content of the current packet is inspected and identified (i.e., video/audio) by the *ADU Identification & Tagging* level. A second layer consists in performing *Rate Control* which should be enforced by any communication protocol not to cause network collapse by congestion. Finally, a third layer performs *Network service Classification & Marking* allowing the protocol to fully take advantage of the QoS mechanisms supported by the network.

In what follows, for illustration purposes we define three sets of modules as our universe D. $D_1 = \{M_1\}$, $D_2 = \{M_2, M_{2'}, M_{2''}\}$, $D_3 = \{M_3, M_{3'}\}$. The functionalities of each of these modules are presented in the following sections.

4.2.1.1 Module's functional presentation

4.2.1.1.1 L_1: ADU Identification & Tagging (M_1) At this level, module M_1 is a *QoS Optimizer*. It is aimed at optimizing the QoS perceived by the user for multimedia flows by taking into account both the level of congestion in the network as well as the QoS properties of the transported flows. This mechanism operates by selectively discarding certain ADUs (Application Data Units) when they are obsolete (i.e., will arrive too late) or of lower priority when the network congestion does not allow for complete transmission of the flow without unacceptable delay accumulation. The algorithm that governs this module's execution is presented in algorithm 4 below.

4.2.1.1.2 L_2: RateControl ($M_2, M_{2'}, M_{2''}$) The *RateControl* level performs traffic shaping in response to congestion. For this, many modules are available performing rate control following different

Algorithm 4 QoS optimizer module algorithm in pseudo-code

Require:
The current ADU $currentADU$.
A reference to the rate control Micro-Protocol $rateControl$.

```
1:  if canBeSent(currentADU, rateControl.sendingRate) then
2:      if currentADU.isHighestPriority() then
3:          send(currentADU);
4:      else
5:          if allCanBeSent(rateControl.sendingRate) then
6:              send(currentADU);
7:          else
8:              drop(currentADU);
9:          end if
10:     end if
11: else
12:     if currentADU.isHighestPriority() then
13:         send(currentADU);
14:     else
15:         drop(currentADU);
16:     end if
17: end if
```

congestion control algorithms such as TFRC (M_2) *(TCP Friendly Rate Control)* [M-13], AIMD ($M_{2'}$) (TCP's standard congestion control) or MULTFRC ($M_{2''}$) (TFRC variant optimized for wireless environments) [M-7].

4.2.1.1.3 L_3: Network service Classification & Marking ($M_3, M_{3'}$) At the *Network service Classification & Marking* level, various modules whose role is to allow the transport protocol to take advantage of the underlying QoS services such as DiffServ or IEEE 802.11e are available. The *QoS Adapter* module (M_3) is specifically designed for IEEE 802.11e environments. This standard defines four categories for channel access. Each of these categories has a different priority in accessing the medium so that high priority traffic be privileged compared to best effort/background traffic. The algorithm for the QoS Adapter module is provided in algorithm 5 below.

The *DS Classifier* module ($M_{3'}$) is specifically designed for DiffServ environments. It allows for marking the packets of a specific application according to the traffic class that has been specified by the user.

Algorithm 5 QoS Adapter module algorithm in pseudo-code

Require:
The current ADU $currentADU$.
An helper object for managing IEEE802.11e classification (80211e)

```
1: if isAudio(currentADU) then
2:     80211e.setAccessCat(AUDIO, currentADU);
3: else
4:     if isVideo(currentADU) then
5:         80211e.setAccessCat(VIDEO, currentADU);
6:     end if
7: else
8:     80211e.setAccessCat(STANDARD, currentADU);
9: end if
```

4.2.1.2 Module's Analytical Presentation

During its execution, M_1 (*QoSOptimizer*) requires access to the value of the variable named *maxSendingRate* $(M_i^P)_2$ and publishes the variable named *frameType* $(M_i^P)_1$. All modules of the L_2 level (*RateControl*) publish the *maxSendingRate* $(M_i^P)_2$ variable. The M_3 module (*QoSAdapter*) requires the knowledge of the *frameType* $(M_i^P)_1$ variable and publishes the *availableACs* $(M_i^P)_3$ variable which corresponds to the access categories supported by the underlying network. Finally, the $M_{3'}$ module (*DS Classifier*) has no publication but requires the *resClass* $(M_i^R)_4$ variable. Following this description, it is possible to define the following matrices where each row respectively represents *frameType*, *maxSendingRate*, *availableACs*, *resClass*:

$$\mathbf{M_i} = \begin{pmatrix} \text{frameType} \\ \text{maxSendingRate} \\ \text{availableACs} \\ \text{resClass} \end{pmatrix}$$

$$\mathbf{M_1^P} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \mathbf{M^P_{2,2',2''}} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \mathbf{M_3^P} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad \mathbf{M_{3'}^P} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\mathbf{M_1^R} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \mathbf{M^R_{2,2',2''}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \mathbf{M_3^R} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \mathbf{M_{3'}^R} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Based on these matrices and by applying the model presented in section 3.3.2, the set of initial possible compositions: M' containing $\sum_{i=1}^{3} \binom{6}{i} = 41$ elements can be reduced by applying (3.1) to $M = \{M_2,\ M_{2'},\ M_{2''},\ M_1M_2, M_1M_{2'},\ M_1M_{2''},\ M_1M_2M_3,\ M_1M_{2'}M_3,\ M_1M_{2''}M_3\}$ in which

only the 9 *valid* compositions appear.

Given no restrictions on the order that these compositions need to be instantiated nor incompatibilities between each other, the resulting fully connected semiautomaton is provided on figure 4.2.

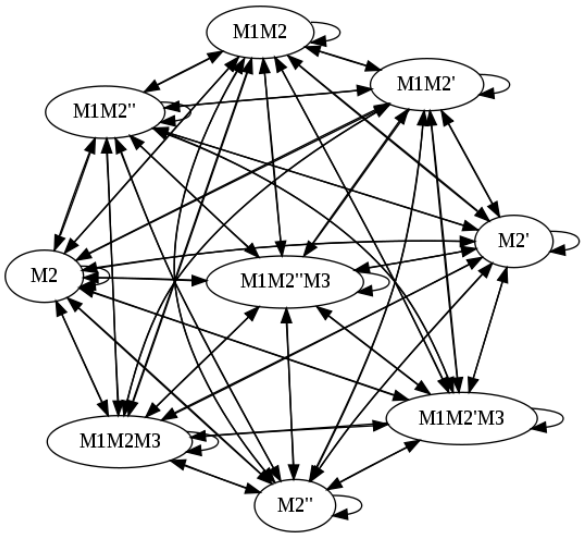


Figure 4.2: Semiautomaton obtained from the nine possible compositions

4.2.2 Simulated environments

4.2.2.1 Experiments

To show the decision model in action, the different *valid* module compositions in M are evaluated in the context of a wireless network following the IEEE 802.11e standard. The performances of these different compositions are evaluated to manually construct the *history* following the algorithms presented in chapter 3.

4.2.2.1.1 Test Platform Description

The compositions in M are evaluated using the architecture presented on figure 4.3 which simulates the transmission of a multimedia flow on an IEEE 802.11e enabled network. The *Stack* element is successively composed by the different compositions of M. However, to fully demonstrate the architecture, the M_2, $M_{2'}$ and $M_{2''}$ services are implemented by a single autonomic Micro-Protocol

which chooses the best algorithm based on the nature of the environment in which it evolves and the nature of the stream it transports (i.e., $M_{2'}$ for wired file transfer, M_2 for wired interactive multimedia session and $M_{2''}$ for wireless interactive multimedia). In all the considered scenarios, the autonomic Micro-Protocol will always choose $M_{2''}$ for congestion control. This further reduces the state space from nine to three possible compositions.

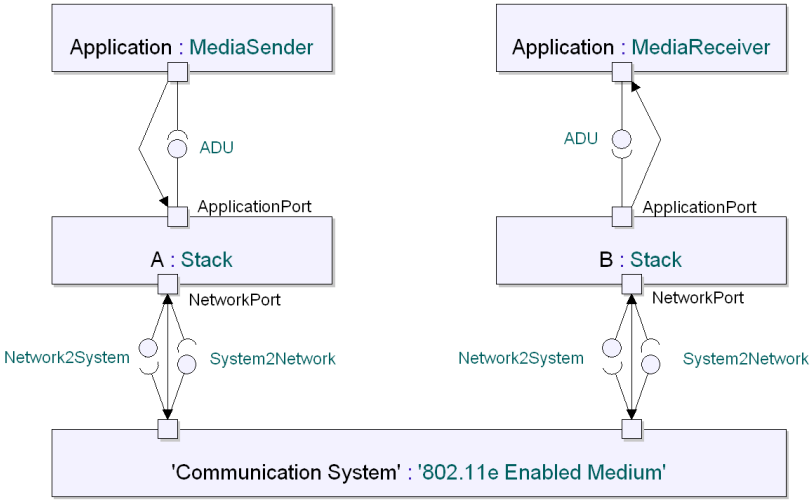


Figure 4.3: Considered experimentation architecture

To evaluate the performances of these compositions, both network metrics (packet loss rate, throughput, delay) and applicative metrics (packet loss rate, delay, throughput) are used. As the stream that is being transported in these scenarios is a multimedia stream, a quantitative evaluation of the quality of the received stream is provided by the value of the PSNR metric.

Peak Signal Noise Ratio

The PSNR is a metric to quantify the quality of a video presentation. It is defined as follows.

Let A and B the luminance matrices of two $(p \times q)$ video images corresponding to the same instant on respectively the original stream (A) and the received stream (B). Given Y_{max} the maximum luminance value (depending on its binary representation, generally 255), the PSNR is defined by equation (4.1) below:

$$PSNR = 10log_{10}\left(\frac{Y_{max}^2}{\frac{1}{pq}\sum_{i=0}^{p-1}\sum_{j=0}^{q-1}(A_{ij}-B_{ij})^2}\right) \quad (4.1)$$

This shows that higher PSNR values correspond to less degraded images than lower PSNR values. When evaluating a video instead of a single frame, the average value of the PSNR computed over all frames can be used as a measure of the overall quality of the video presentation. However, for finer grained analysis, either a per frame or sliding window average graph of the PSNR values can be used.

IEEE 802.11e Model

The IEEE 802.11e medium simulation is based on the performance analysis conducted in [M-15]. This study shows the existence of a maximum attainable throughput for each access category when the system is evolving under saturation (i.e., has at least one packet to be sent in each queues at all times).

In all the scenarios that follow, the network will be considered as being saturated by video and best effort traffic such that the maximum attainable throughput for an addition flow is of 150Kbps in the best effort category and 300Kbps in the video category.

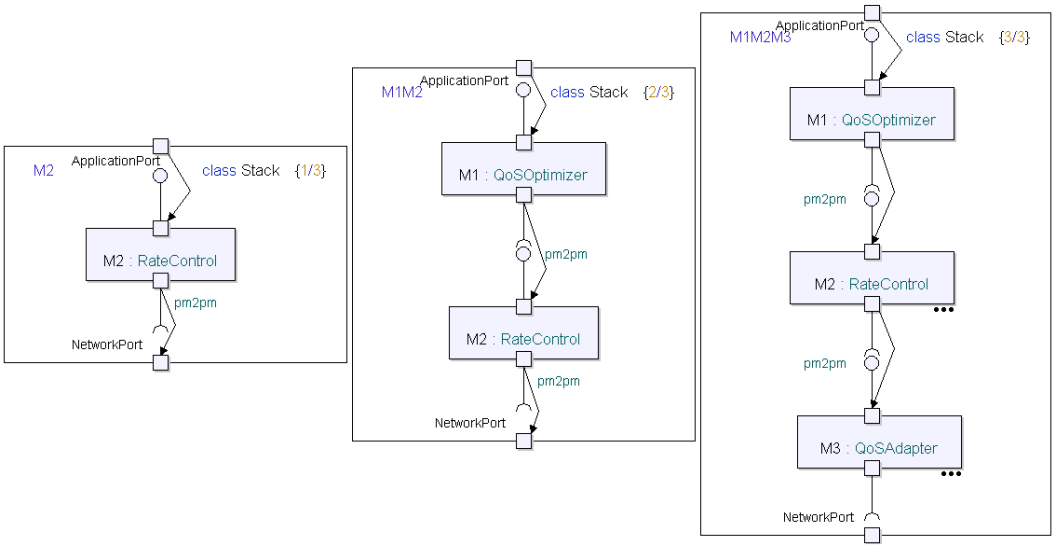


Figure 4.4: Test Scenarios : All possible compositions for *Stack*

4.2.2.1.2 Test Scenarios Presentation

The different case studies are synthesized by figure 4.4. The goal is to evaluate each of the possible solutions for composition to construct the MDP that will be used for decision making in future executions of the agent in these environments.

For every scenario above, an H.263 encoded video using the CIF frame size is used. The traffic profile of this video is presented on figure 4.5. The ACT is provided with the goals $L_{max} = 30\%$, $T_{min} = 750Kbps$ and $D_{max} = 50ms$. These goals could be the ones expressed by an application performing video conferencing using a highly redundant codec.

4.2.2.2 Results

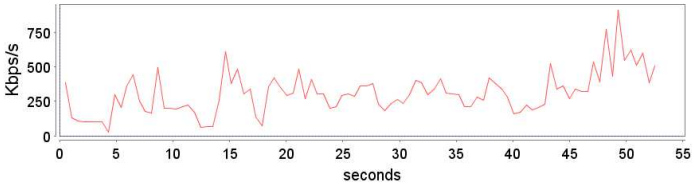
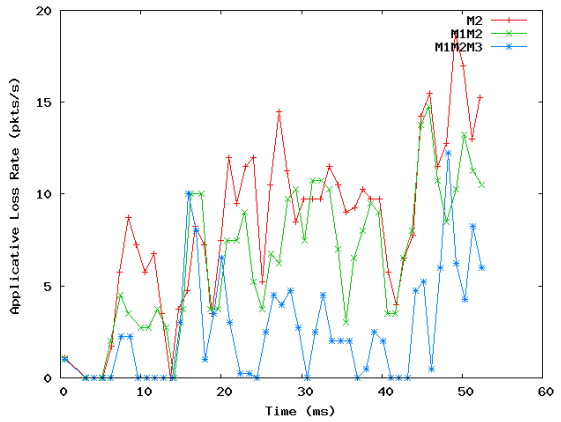


Figure 4.5: Traffic profile of the considered video

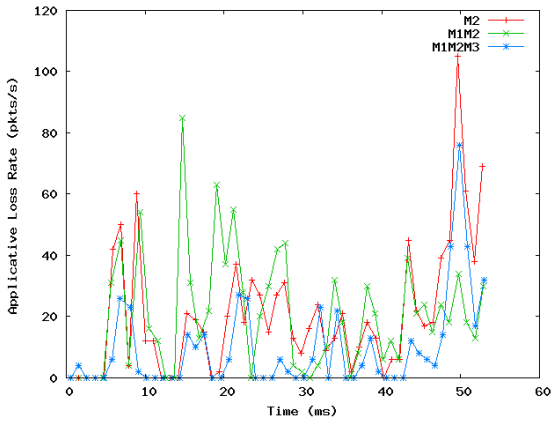
The results obtained for the different compositions in the simulated environment are presented on figure 4.6 and summarized in table 4.1 below. As the simulation environment does not support the measurement of delays (intrinsic limitation of the TAU simulator), only throughput and loss rate are evaluated. Although this limitation reduces the spectrum of tests that can be performed using the simulator, it does not impact the functional validation of this proof of concept.

Table 4.1: Results for each considered composition

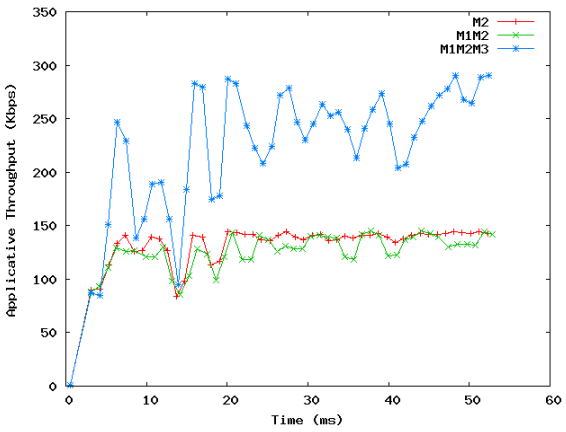
Composition	Global LR (pkt/s)	LR I Images Subflow (pkt/s)	LR P/B Images Subflow (pkt/s)	Average Throughput (Kbps)	Average PSNR (dB)
$M_{2''}$	27.39	8.59	18.80	133.68	12,38
$M_1M_{2''}$	28.51	6.63	21.88	126.91	14.67
$M_1M_{2''}M_3$	12.97	2.76	10.21	227.74	21.11



(a) Packet Loss Rate (I frames subflow)



(b) Packet Loss Rate (P/B frames subflow)



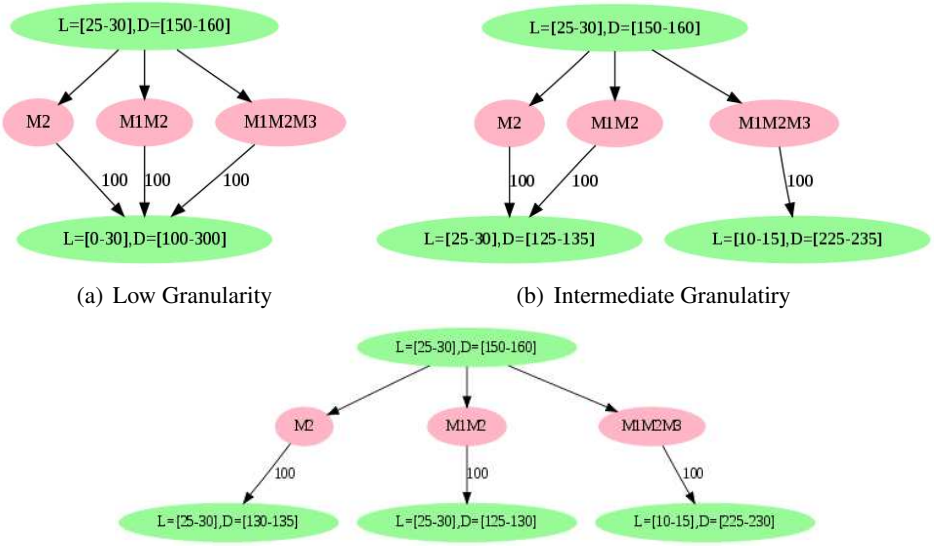
(c) Throughput (Kbps)

Figure 4.6: Simulation Results

In this simulation, the environment state does not change as it is a parameter of the scenario. The ACT definition of the environment state provided during the ACT design (see section 2.3.3.3) consists in evaluating the service obtained to an application when no modules are instantiated. The simulation parameters are set so the service obtained when using no composition is characterized by $L \in [25..30]\ pkts/s, T \in [150..160]\ Kbps$.

The monitoring of the service characteristics obtained with each composition allows to build the partial MDP in which only one state is considered for the environment. In this environment state, the agent can decide to instantiate any composition among $M_2, M_1M_2\ or\ M_1M_2M_3$.

The TAU G2 simulation tool used in these tests is deterministic. By such, the service char-



(a) Low Granularity

(b) Intermediate Granulatiry

(c) High Granularity

Figure 4.7: Simulation Results

acteristics obtained in the environment are identical with each simulation. The uncertainty on the environment state and service characterization can therefore not be taken into account in this example and the constructed MDP is fully deterministic.

Depending on the quantization granularity, the services resulting from these compositions can either be seen as one, two or three different states, as presented on figure 4.7. However, the one state MDP is more unlikely as it covers huge sections of the service characteristics state space. Given the learned MDP, the following transmissions in similar environments will use the composition $M1M2M3$ which has shown to provide better results by bringing the service characteristics closer to the application goals.

Once all possible actions have been explored, the MDP is complete and the learning algorithm will only monitor the performance of the services resulting from the decision algorithm to capture possible changes and adapt the MDP accordingly.

Figure 4.8 presents measures of the effective quality of the service provided to the applications. This quality is measured using the PSNR metric which is more specific to video streams than to networking. The graph shows the evolution of the average PSNR value computed by the receiver every

second. The results however are encouraging in the sense that the mean PSNR value for the scenario in which composition $M_1M_2M_3$ is instantiated is much higher than with the two other compositions. This observation can be correlated with the service characterizations. Indeed, the service provided by M_1M_2 is very close to the service provided by M_2 while the $M_1M_2M_3$ composition clearly provides a superior service.

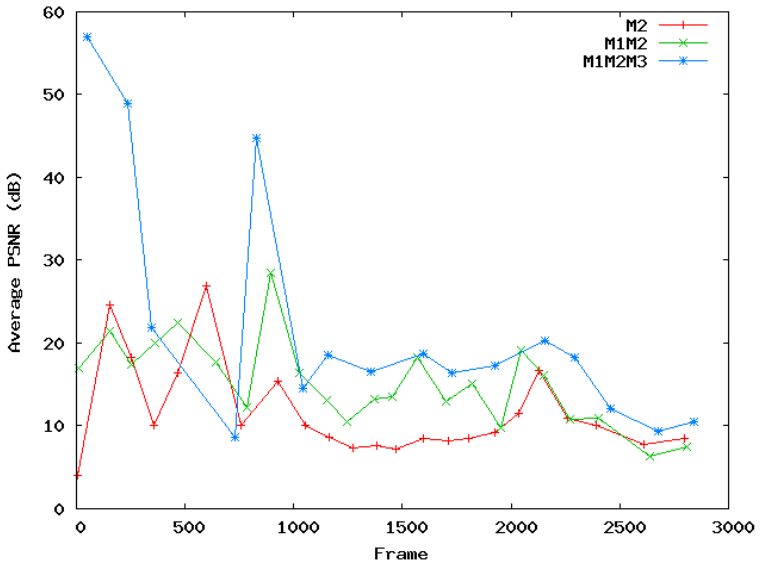


Figure 4.8: PSNR comparison between compositions (PSNR average computed every second)

From the above results, it appears that the decision provided by the decision algorithm given the constructed MDP, $M_1M_2M_3$ corresponds to the best possible choice among the different compositions of M.

4.3 Autonomic Micro-Protocols

To demonstrate the Autonomic Micro-Protocol approach in the ETP framework (ATP), the following paragraphs present experiments performed for given application specified goal under different network scenarios. After presenting these scenarios, the different graphs are detailed and the results are commented.

4.3.1 Test scenarios

Three scenarios are used to demonstrate the benefits of the ATP over statically defined protocol composition and parameter values. In the considered scenarios, algorithm 3 presented in section 3.4.3 is used for the control loop.

For all scenarios, the application specified QoS goals are as follows:

- Max Delay: 50ms
- Max Loss Rate: 30%
- "Low Delay" has priority over "Low Losses"

The network conditions are such that in all scenarios, the network RTT is set to be 20ms and losses are of random nature following a uniform distribution. The loss rate varies depending on the scenario, it is respectively set to: 40%, 50% and 60% for scenarios I, II and III. All these parameters are summarized on table 4.2. Although these values might seem very high and far from most practical environments, they are chosen to illustrate the feasibility of the approach even in highly unreliable networks.

Table 4.2: Test Scenarios parameters

Scenario	Network LR	App. Target LR	Network RTT	App. Target Delay
I	40%	<30%	20 ms	<50ms
II	50%	<30%	20 ms	<50ms
III	60%	<30%	20 ms	<50ms

For each scenario, the results are compared to those obtained using a protocol which is incapable of taking the application requirement into account nor adapt its parameters. In this case, we use the ETP composed by a fully reliable SACK based error control mechanism.

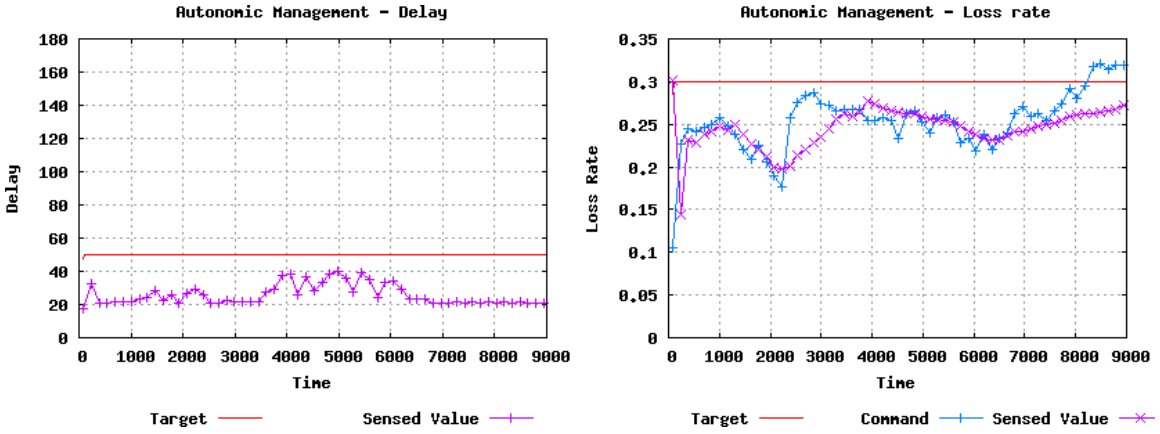
4.3.2 Results

This section presents the results obtained when using the ATP for the above described scenarios. These results are compared to those obtained when using the ETP with a single Micro-Protocol in-

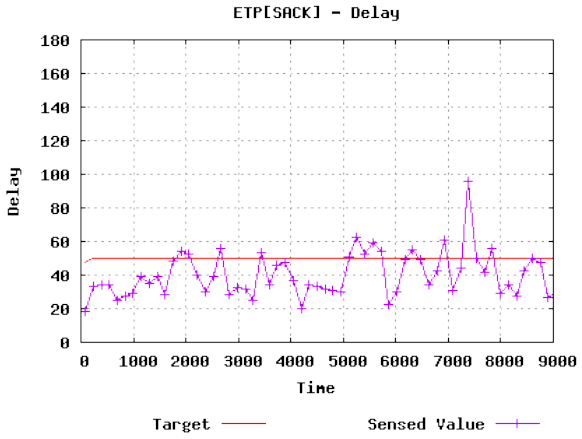
stantiated performing fully reliable selective acknowledgment (SACK) based ARQ (Automatic Repeat reQuest).

4.3.2.1 Scenario I

The results obtained for scenario I are presented on figure 4.9(a) when using the ATP. The results obtained for ETP in the same environment are presented on figure 4.9(b).



(a) ATP



(b) ETP with SACK-FR

Figure 4.9: Scenario I

The graph presented on figure 4.9(b) shows the delay for the data packets to be delivered to the application running on the receiver. In this scenario, the ETP modular transport protocol is used

with only a selective acknowledgment (SACK) based ARQ reliability Micro-Protocol. Moreover, the fact that ETP is not capable of self adaptation makes it so the reliability parameter of this ARQ Micro-Protocol to always remain 100% meaning that it will never tolerate any loss and always try to retransmit lost packets. In this scenario the loss rate is by such always null.

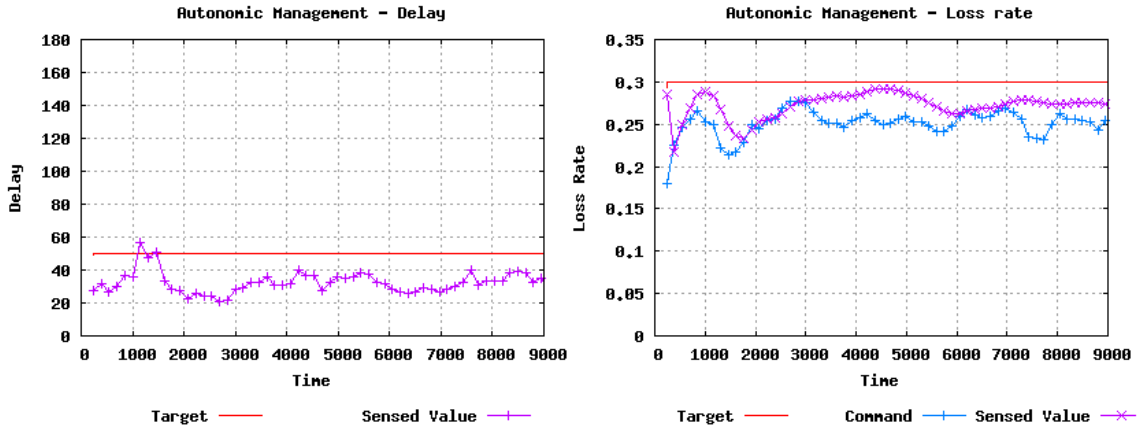
The horizontal red line on the graph denotes the application's QoS limit. All data for which the delivery delay exceeds 50ms cannot be used by the application (for example due to some real time constraints). From what can be seen on the graph, the ETP[SACK] composition is not able to maintain the delay under the application's threshold and many packets arrive with a delay greater than this limit.

When examining the performances of the ATP in a similar context, figure 4.9(a) shows both the applicative delay (left part) and the applicative loss rate (right part). On the delay graph, it can be seen that the delivery delay is kept below the applicative threshold at all times meaning that the application will be able to make use of all the data that is being delivered to it.

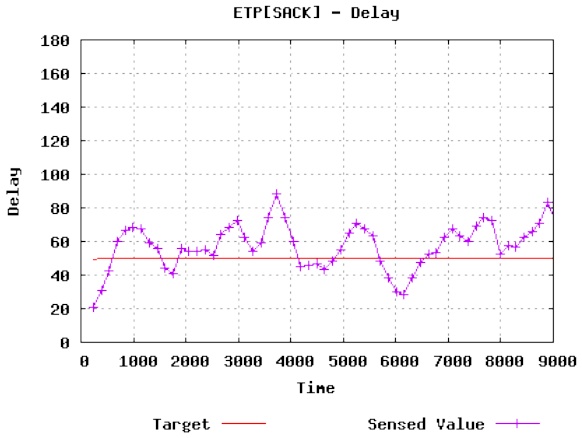
A further study of the ATP behavior shows that, to be able to always guarantee such low delays, the ATP has instructed the ARQ Micro-Protocol to tolerate different loss rates during execution. However, it can be seen that the sensed value of the loss rate (presented on the right part of figure 4.9(a)) always remains below the maximum value tolerated by the application. While examining the command value that is computed by the ATP Management module, it can be seen that it changes depending on the network condition to always maintain the lowest possible loss rate as long as the delay conditions are satisfied.

4.3.2.2 Scenario II

In the second scenario, the applicative goals for both delivery delay and packet loss rate remain unchanged. However, the network conditions change in the sense that the emulated network packet loss rate is now 50%. This high loss ratio makes it such that half of the packets need at least one retransmission by the ARQ mechanism to reach the receiver. Moreover, the average transmission delay becomes 30ms which is close to the application specified limit of 50ms.



(a) ATP



(b) ETP with SACK-FR

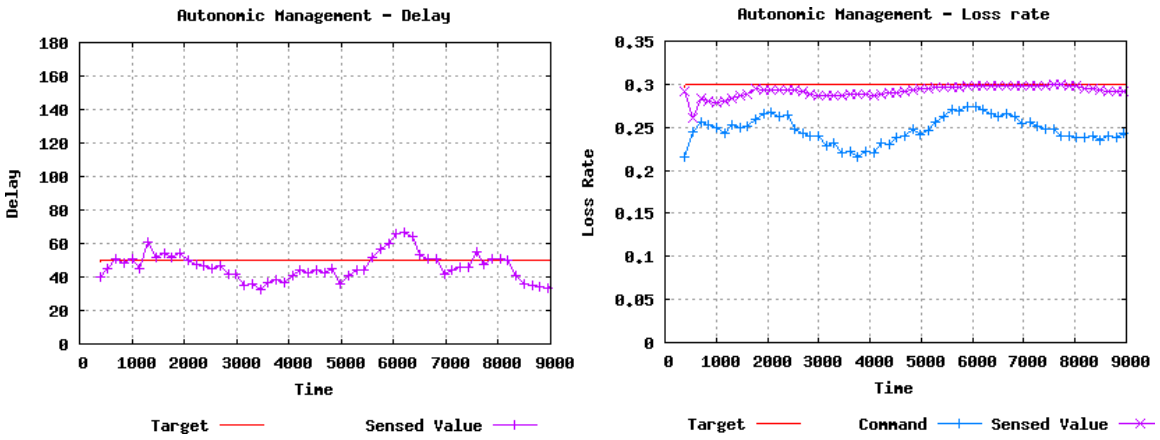
Figure 4.10: Scenario II

In such context, when using the ETP[SACK] non adaptive transport protocol, it can be seen that most of the packets are delivered to the application too late. Indeed, figure 4.10(b) shows that most of the time, the applicative delay is above the application's target resulting in this data being useless to the application. This is due to the fact that the ETP[SACK] ARQ mechanisms parameter are statically set to provide full reliability. The ETP[SACK] completely ignores the fact that the application can tolerate losses of up to 30%, resulting in under-optimal performances.

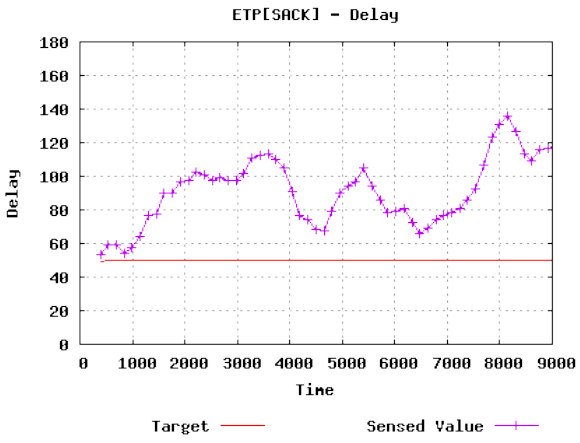
When compared to the results obtained by using the ATP presented on figure 4.10(a), it can be seen that the delay goals are satisfied most of the time. Moreover, the losses goals are also met (right part of figure 4.10(a)) at all times.

The first part of the graph (between times 1000 and 1500) shows a violation of the applicative targets for the delay parameter. However, the command is quickly adjusted to find a good trade-off between losses and delay and the oscillatory behavior that can be observed at the beginning of the connection's life is stabilized.

A second period on the graph between times 4500 and 7000 shows that a decrease in the loss ratio can be achieved without producing an unacceptable increase of the transit delay, the ATP reacts by adjusting its command to try to keep the loss ratio as low as possible while still providing good transit delay to the application.



(a) ATP



(b) ETP with SACK-FR

Figure 4.11: Scenario III

4.3.2.3 Scenario III

For the third scenario, the application QoS goals remain unchanged but the network loss ratio is now set to 60%.

The results obtained when using the ETP[SACK] transport protocol are presented on figure 4.11(b). Indeed, it can be seen that the full reliability parameter of the ETP composition makes it impossible to meet the applicative target delay at any time during the connection resulting in the application being unable to process any of the received data (as it is received too late).

When using the ATP, the graphs on figure 4.11(a) demonstrate that delay requirements are met most of the time. Indeed, the average delivery delay is located very close to the application specified limit, demonstrating that the mechanism is reaching its limits.

On the right part of figure 4.11(a), the ARQ SACK command as well as the sensed applicative loss rates are shown. It can be seen that the delay violations that occur around times 1000 are quickly stabilized as the command is raised accordingly.

The second violations occur around times 5500 and the analysis of the ARQ mechanism command graph (right part of the figure) demonstrates that the mechanism has reached its limit. Indeed, the sensed loss rate has reached the limit specified by the application. The ATP will therefore not issue higher commands not to violate that parameter.

Finally, it can be seen that, around time 6500, the ATP senses improvement on the delay and therefore lowers the value of the ARQ command to reduce the loss rate.

4.4 Learning and adapting in real network environments

The previous sections have shown how both the ACT algorithms and concepts as well as Autonomic Micro-Protocols could be used to dynamically optimize the service provided to the applications. In this section, the java implementation of the ACT is evaluated in a real network environment. On the platform presented in section 4.1.1, an application using the ACT is successively presented with a variety of environments. In a first part, the discovery and construction of the MDP by the learning

algorithms is observed and commented. A second part of this section studies the service provided to the application when network conditions change in a random fashion.

4.4.1 Test scenarios description

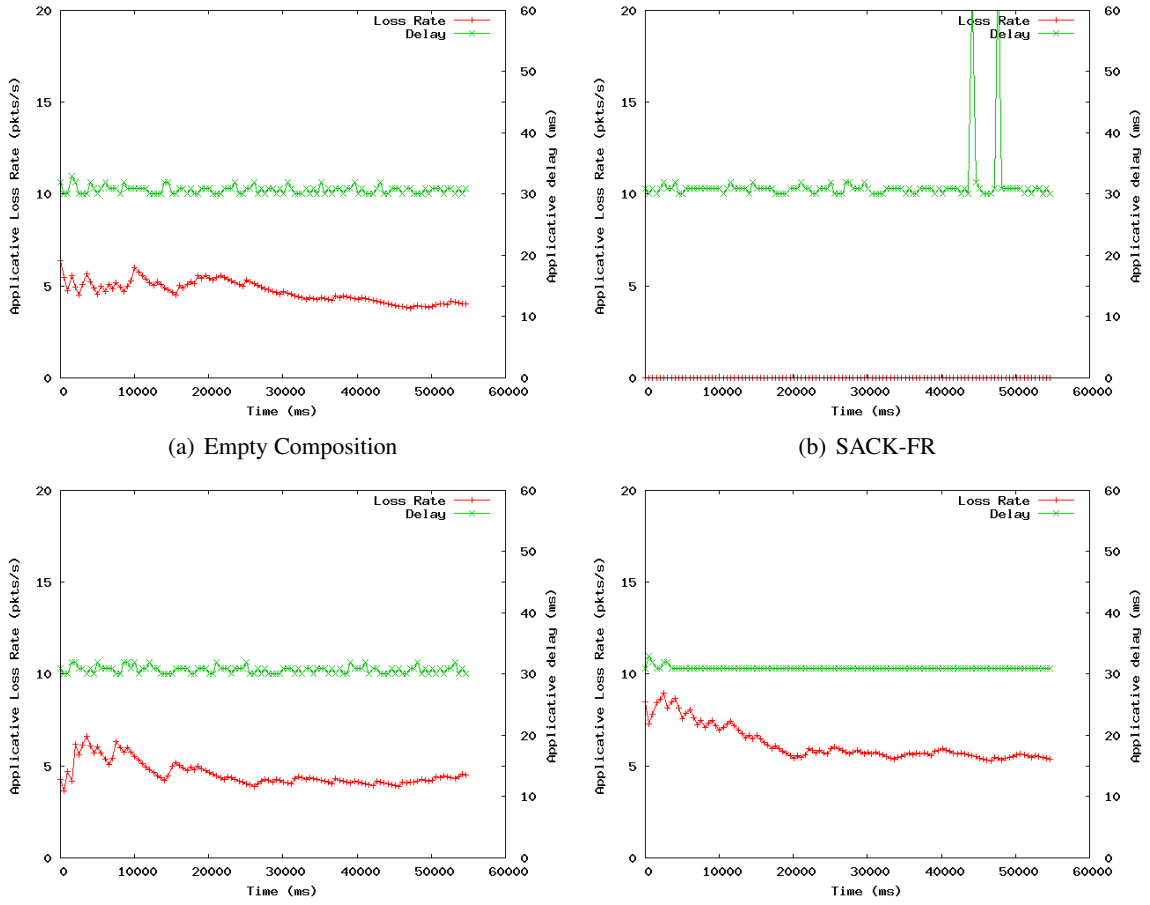
4.4.1.1 Service compositions

The goal of the experiment is to demonstrate more aspects than the proof of concept made through simulation in 4.2 while still being easily observable. In order to achieve this, the universe of Micro-Protocols that can be composed together comprises a fully reliable SACK, a partially reliable autonomic error control and TFRC. Additionally, to limit the size of the semiautomaton for the purpose of illustration, the maximum number of Micro-Protocols that can be composed together is set to one. This simplification affects the construction of the semiautomaton. This construction does not require the composition model to be performed as the graph is fully connected, each one of its four nodes representing one Micro-Protocol. The reduction in the size of the semiautomaton does not trivialize the learning and decision processes, however, it makes them easier to be analysed. This analysis is presented hereafter.

4.4.1.2 Network conditions

In order to observe the learning algorithm at work, in a first part of the experiments, the network conditions are changed every minute until each of the possible combinations of losses and delay is presented 10 times to the ACT. The loss rate varies between 5% and 30% in 5% increments while the delay varies between 10ms and 100ms in 10ms increments.

Under these parameter values, the environment changes 600 times during the 10 hour period that the experiment lasts. This allows for the ACT to test every composition at least twice in every environment state. Every time the ACT detects a change in the environment, it creates a trace file in which it logs all the monitoring values that are used for constructing the MDP. This allows for the process to be observed thoroughly.



(a) Empty Composition

(b) SACK-FR

(c) Autonomic Error Control

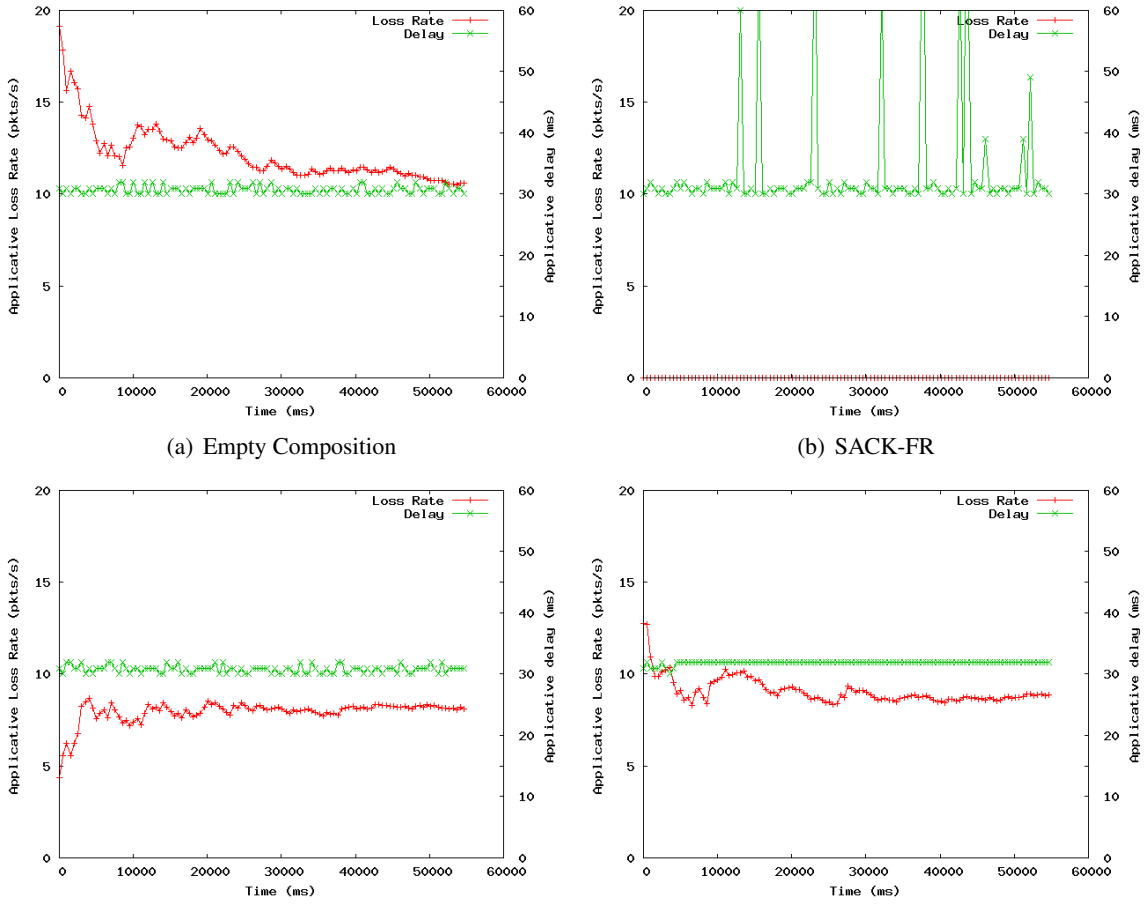
(d) TFRC

Figure 4.12: Agent service Characterization for 5%/30ms environment

4.4.2 Learning: MDP construction

After the experimentation is completed, the agent's learned MDP is comprised of 378 nodes and 720 edges. Of these nodes, 60 represent the environment states, 240 are used to represent the possibility of applying each one of the four compositions in each environment state and 78 represent service characteristics. The average number of service states that can be reached by applying a given composition in a specific environment is of 2. In average there are 6 environment state/composition pairs that lead to a given service state.

Given the high number of nodes and transitions needed to represent the full knowledge of the agent after the experiment (see appendix 4.5.1.2), the detailed analysis of the learning algorithm



(a) Empty Composition

(b) SACK-FR

(c) Autonomic Error Control

(d) TFRC

Figure 4.13: Agent service Characterization for 10%/30ms environment

behavior can only be observed on a subset of the environment states. Moreover, to further reduce the size of the MDPs for the analysis, only losses and delay criteria are taken into account in the experiments. In what follows, the observation in which the agent determined the environment states with Delay equal to 30ms and losses respectively 5% and 10% are considered.

The graphs presented on figures 4.12 and 4.13 present the observations of the service characterization by the agent in one of its execution over each environment for each of the available compositions.

As it can be observed on figure 4.12, the full reliability SACK Micro-Protocol provides a zero-loss service which comes at the price of several uncontrolled increases in the delay. On the other hand, a TFRC congestion control algorithm seems to be the worst choice in terms of losses, mainly

just after the reconfiguration took place.

When the environment loss rate increases, figure 4.13 shows that things are different. The fully reliable mechanism still provides zero-losses but the uncontrolled increases in the delay are much more frequent. Using TFRC or the Autonomic Error Control appears to control the loss ratio in comparison to what is obtained when using an empty composition. Indeed, while the Autonomic Error Control will retransmit data, it will do so to avoid uncontrolled delay increases. In a similar way, the TFRC congestion control algorithm will regulate the flow to minimize (or at least control) losses.

Figure 4.14 presents a graphical representation of the partial MDP built by the agent after the execution that led to the graphs presented on figures 4.12 and 4.13. The nodes are labeled starting with the letter *e* to denote environment state and the letter *s* for service characteristics. Following the letter are two numbers which correspond respectively to the packet loss rate and delay. Finally, some nodes labels have an additional number which is used to denote a given service where 1 is the empty composition, 2 represents SACK-FR, 3 is used for the Autonomic Error Control while 4 is intended to describe TFRC.

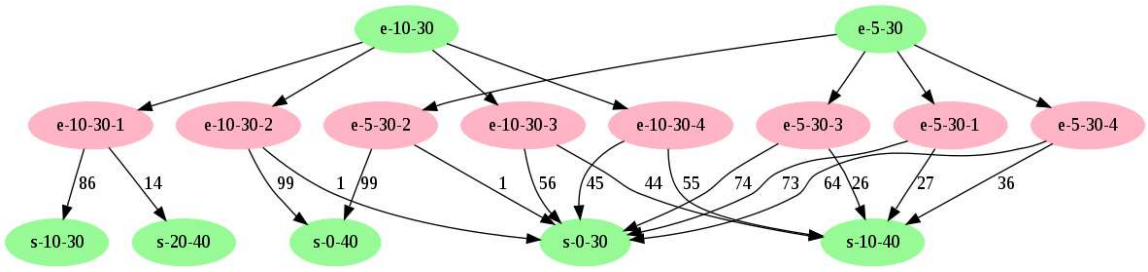


Figure 4.14: Partial representation of the agent's MDP

As it can be seen, the various service compositions can lead to five different services to be provided to the application in the considered environments. Not all services can be reached in all environments and some services such as *s-0-30* can be reached from both environment state by using the Autonomic Error Control or TFRC. In both cases, composition 3 (Autonomic Error Control) has the higher probability of taking the system into *s-0-30* with respectively 56% and 74% in *e-10-30* and *e-5-30*. On the other hand, while it is possible to obtain a service of *s-0-30* when using SACK-FR

in *e-10-30* it is much unlikely as the service resulting from the retransmissions induced by the full reliability lead to a small increase of the delay which is characterized by *s-0-40*.

4.4.3 Full adaptation scenario

To demonstrate the adaptation capabilities of the agent once the MDP has been constructed, the following paragraphs analyse the agent's behavior in varying network conditions. The goal of the agent in what follows consist in maintain a delay below 150ms (i.e., an average recommended value for interactive real time applications such as VoIP) and a loss rate below 10% (i.e., that can be tolerated by using "adaptive coding").

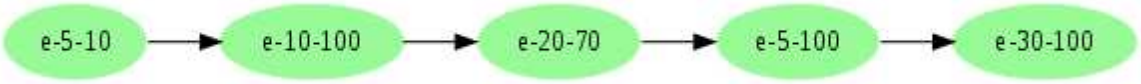


Figure 4.15: Scenario Environment Transitions

In this scenario, the network platform described in 4.1.1 is used to emulate network conditions varying according to the graph presented on figure 4.15. This sequence of environment changes is much unlikely to happen in common executions of the ACT but is presented intentionally to demonstrate that the approach is not limited to slightly changing network environments and also works well in highly dynamic contexts. The change frequency is set so that a change will occur every 60 seconds. On the graph, node labels follow the same syntax as the ones on figure 4.14 (i.e., *e-lossRate-Delay*).

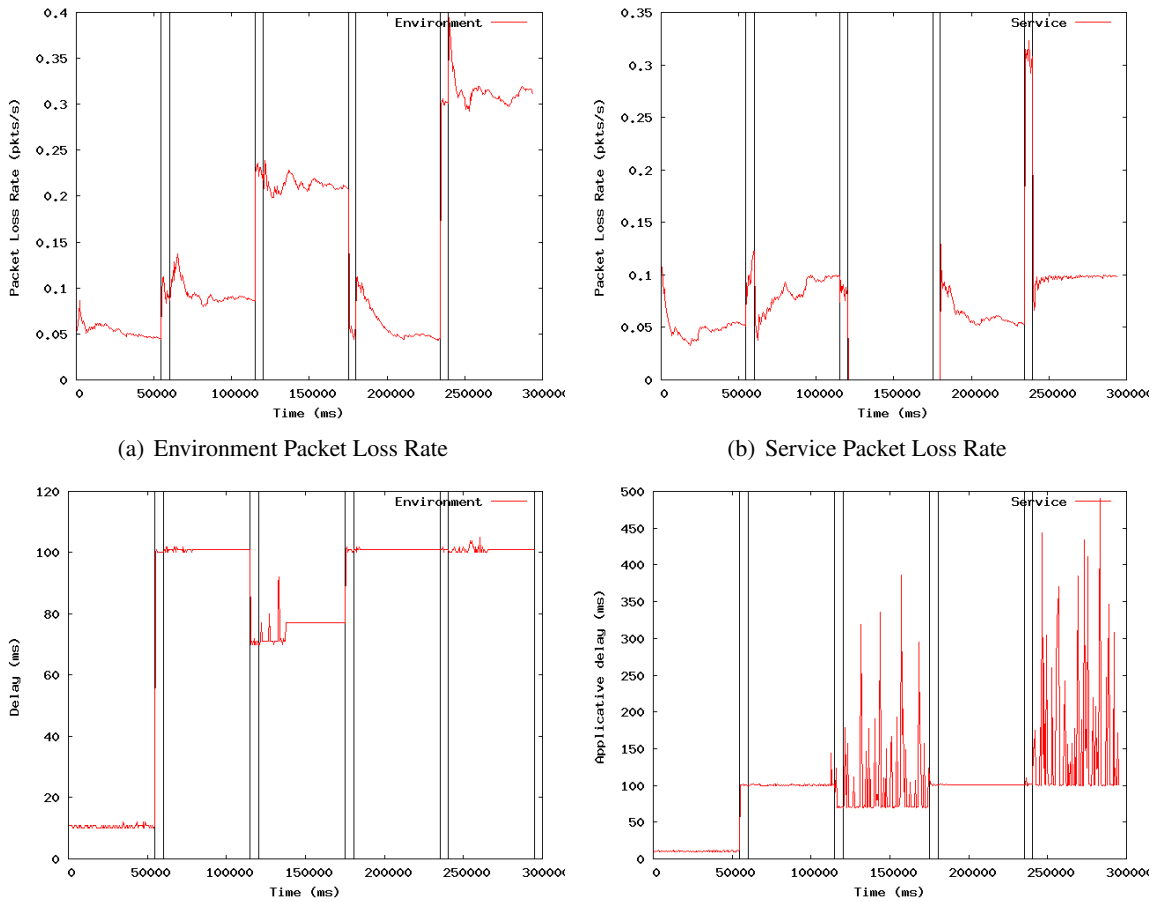
During the scenario execution, the agent has knowledge of the full MDP obtained during the learning process presented in section 4.4.2. Following the ongoing learning and monitoring algorithms, the present scenario will lead to the MDP to be further refined based on the new experience gathered during its execution. Although the full MDP is too big for its graph to be analysed, the information required to analyse the behavior of the agent in its present execution have been extracted and will be referenced when required below.

Figure 4.16 presents a graphical representation of the agent's monitoring of both the environment (figures 4.16(a) and 4.16(c)) and the service provided to the application at all times (figures 4.16(b) and 4.16(d)). On the graphs, both the moment at which the agent detects a change to the environment

which could lead to its reconfiguration should further measurements present similar values and the moment at which the agent decides to reconfigure itself are represented by vertical lines.

The first environment change is perceived by the agent as an increase in both the loss rate and the delay. Following this change, the agent uses the MDP in order to find the solution which best suits the application goals in the new environment state. The partial representation of the agent's MDP presenting only the information that is relevant to the states which are considered in the present scenario are presented on figure 4.17.

When looking at figure 4.17(a), it can be seen that the best choice to obtain a service which maximizes the agent's reward computed by the $\gamma()$ service efficiency function presented in section



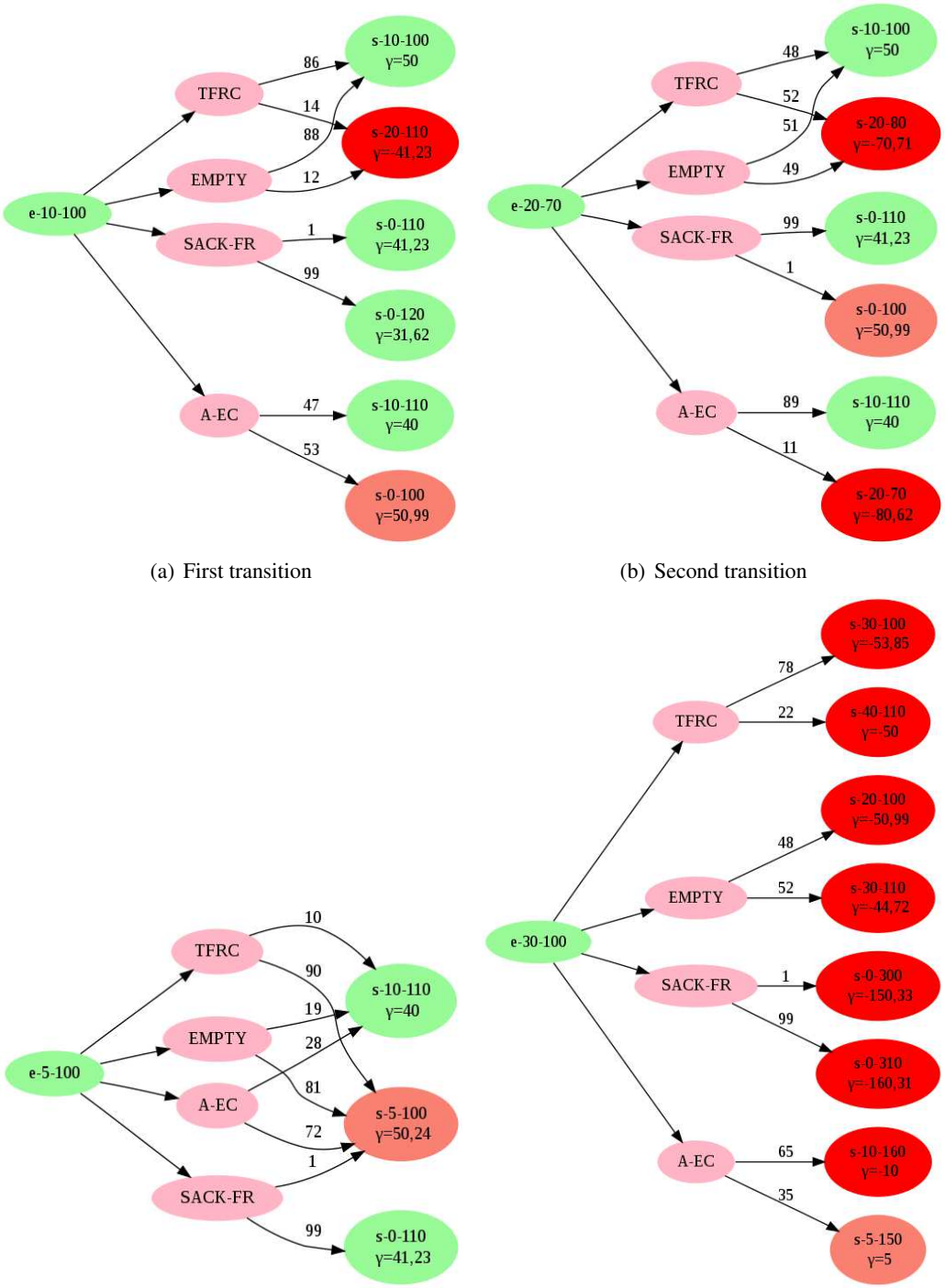
(a) Environment Packet Loss Rate

(b) Service Packet Loss Rate

(c) Environment Delay

(d) Service Delay

Figure 4.16: Environment and service Monitoring in full scenario



(a) First transition

(b) Second transition

(c) Third transition

(d) Fourth transition

Figure 4.17: Partial MDPs for scenario transitions

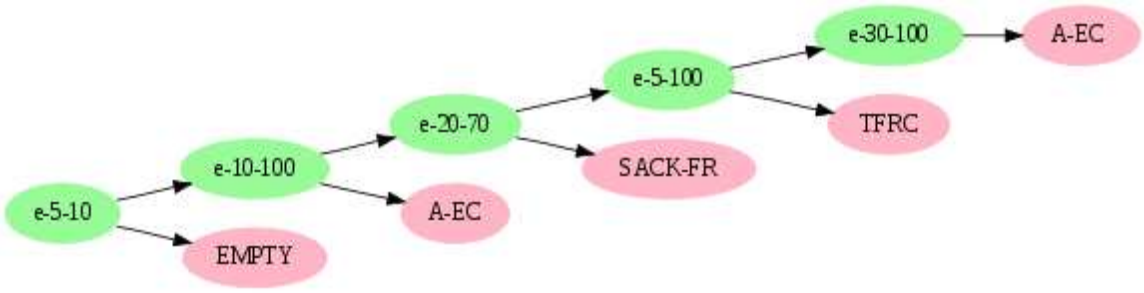


Figure 4.18: Scenario Environment Transitions

2.3.3.4 corresponds to instantiating the Autonomic Error Control Micro-Protocol which will take the system into the best service state (*s-0-100* with a probability of 53%) and does not lead to any state in which γ is negative. The next transition decision is straightforward as the service can be maintained with a 99% probability and maybe even improved by applying the fully reliable SACK vector based error control. In the third transition, TFRC seems the best option as it leads to best possible performances with 90% probability. Finally, the fourth transition re-instantiates the Autonomic Error Control which has the highest score and constitutes the only solution in this context. However, the choice of the A-EC module might also lead to a suboptimal state in which γ is negative. In this case, if the environment does not change, the ATP will have reached the limits of its adaptation capacity and will notify the ASM for a higher level solution to be searched for.

Figure 4.18 provides a synthetic view of the choices made by the agent in this scenario's execution. As illustrated, these choices totally correspond to the result of the decision algorithms when applied to the partial MDPs as described in the previous paragraph.

From the above study, it can be seen that the prototype implementation of the ACT behaves as expected. Indeed, after a certain amount of time throughout which the agent performs learning, the system converges to an MDP which can be used to efficiently perform adaptation. As it has been shown, the system always maintains adequate service characteristics for the application. The following sections study the scalability of the approach at the Transport layer.

4.5 Scalability and costs analysis

To estimate the scalability of the approach as well as the costs involved in the dynamic reconfiguration of the communication stack, the following section presents several metrics and scenarios which illustrate different aspects of the approach.

4.5.1 Scenario description

In this section, the various scenarios that have been taken into account in the cost analysis are presented. These tests have been performed by using a java implementation of the enhanced transport protocol which has been enhanced to be able to dynamically change the composition during the communication.

4.5.1.1 Local stack reconfiguration

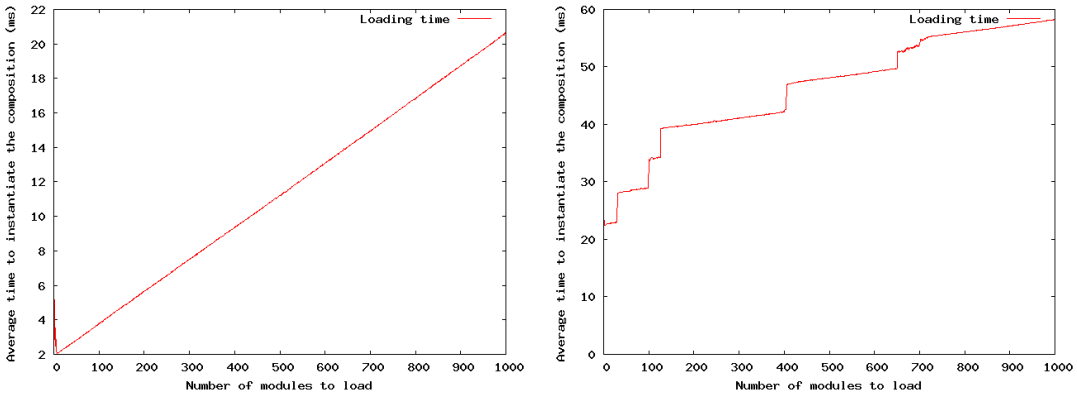
In this first scenario, adaptation is purely local and requires no interactions with the adapting peers. In this context, the enhanced transport protocol implementation is successively asked to load several Micro-Protocol compositions. Each one of these compositions contains an increasing number of

Algorithm 6 Local Stack Reconfiguration

Require:

An instance of an ETP kernel's dynamic loader dl.
A function that creates a composition file with a varying number of modules $createComposition(intnumModules)$.
A clock providing the current time, now.

```
1: emptyComposition ← createComposition(0)
2: for i = 0 to 1000 do
3:     currentComposition ← createComposition(i)
4:     timeStart ← now
5:     for j = 0 to 1000 do
6:         dl.reconfigure(currentComposition)
7:     end for
8:     timeEnd ← now
9:     save(i + " " + timeEnd-timeStart)
10: end for
```



(a) Results for the local adaptation scenario

(b) Results for the distributed adaptation scenario

Figure 4.19: Stack reconfiguration benchmark

processing modules to be instantiated. The algorithm for this test is presented in pseudo-code notation in algorithm 6.

The results obtained for the local adaptation scenario are presented on figure 4.19(a). On the graph, it can be seen that the delay to complete the reconfiguration increases linearly with the number of modules to be replaced. By performing a first order linear regression, the following relation can be obtained: $T = 0,018547 \times N$ where T is the average time it takes to load a composition of N modules into the kernel. The average time to load one module is by such 0,018 milliseconds.

The vast majority of composition in ETP are smaller than 10 modules which makes the local reconfiguration time negligible.

4.5.1.2 Distributed stack reconfiguration

A second scenario consists in performing coordinated adaptation between two adapting peers. In this configuration, one of the peers is chosen to be the master which will decide on the composition to be applied and the other peer is considered to be a slave as it only obeys the master.

In this scenario, both peers are considered to have all the modules they need available locally. Moreover, adaptation might require peers to reach a point where it can safely take place such as a synchronization point. In the present study, both peers are supposed to reach the synchronization point immediately with no delay.

The algorithms that were used for this test are presented in algorithms 7 and 8 for the Master and the Slave respectively.

Algorithm 7 Distributed Stack Reconfiguration - Master

Require:

An instance of an ETP kernel's dynamic loader dl.
A function that creates a composition file with a varying number of modules $createComposition(intnumModules)$.
A clock providing the current time, now.
A socket to communicate with the slave s.

```
1:  emptyComposition ← createComposition(0)
2:  for i = 0 to 1000 do
3:      currentComposition ← createComposition(i)
4:      timeStart ← now
5:      for j = 0 to 1000 do
6:          s.connect()
7:          s.send(currentComposition)
8:          dl.reconfigure(currentComposition)
9:          s.waitForAck()
10:         s.close()
11:         s.connect()
12:         s.send(emptyComposition)
13:         dl.reconfigure(emptyComposition)
14:         s.waitForAck()
15:         s.close()
16:     end for
17:     timeEnd ← now
18:     save(i + " " + timeEnd-timeStart)
19: end for
```

Algorithm 8 Distributed Stack Reconfiguration - Slave

Require:

An instance of an ETP kernel's dynamic loader dl.
A socket to communicate with the master s.

```
1: loop
2:     s.accept()
3:     currentComposition ← s.readComposition()
4:     dl.reconfigure(currentComposition)
5:     s.sendAck()
6:     s.waitForClose()
7: end loop
```

The results obtained in the distributed scenario are presented on figure 4.19(b). It can be seen that

the fact that the adaptation protocol requires a complete round trip of exchanges per reconfiguration translates in an incompressible minimal value for an iteration of around 20ms. Furthermore, when compared to the local adaptation graph, it can be seen that the graph is linear only by parts. However, similarly to what was observed in the local adaptation scenario, the time required to load a typical composition composed of less than 100 modules is still relatively small with a maximum value of 30ms.

Based on this scalability study, it can be concluded that the action that consists in reconfiguring the transport entities taking part in a given communication does not present any limitation to the adaptation strategies presented in this manuscript. Indeed, the reconfiguration of a transport protocol stack is only required when the environment changes which doesn't happen as often as every 30ms.

Conclusion

In this chapter, the concepts which are the base to the ACT architecture have been studied and implemented in simulation (using the TAU G2 modeling tool following an MDA approach) as well as in emulated network environments (using the Linux Network Emulator framework).

The simulation studies have illustrated the concepts and algorithms that allow the system to learn on a simple example. This proof of concept demonstrated the use of the algorithms presented in chapter 3 for constructing the state space and building the semiautomaton on which the decision and learning processes rely. The execution of the ACT lead to the construction of simple MDPs in which knowledge was stored. The importance of granularity has been demonstrated and its direct impact on the quality of decisions has been illustrated.

A larger scale study has been conducted in the emulated network environment, both the Autonomic Micro-Protocol approach and the Transport prototype implementation of the ACT have been evaluated. The results obtained in the various experiments have confirmed that the approach allows to bring autonomy to the complex task of service composition for QoS in networks.

The layering of the decision process by separating the global problem into simpler local subproblems has shown to be beneficial as it allows for the logic of the components to be kept simple.

For instance, the performance observed when using autonomic Micro-Protocols to manage parameter values through simple algorithms illustrate this view.

The decision and learning models have been demonstrated for a broad range of emulated network environments. After a first learning period which was used to bootstrap the agent's knowledge, its execution over changing network conditions following a specific scenario has shown that the coordination of behavioral and architectural adaptation allows to maintain acceptable services for the application even in adverse, highly unstable network situations.

Finally, to demonstrate the scalability of the approach, reconfiguration measurements performed both in a local and distributed fashion have shown that the dynamic reconfiguration of communication entities did not present a bottleneck for performance.

GENERAL CONCLUSION

This thesis has presented the ACT (Autonomic Communication Toolkit), a framework of concepts, models and algorithms for overcoming the limitations of present communication systems in terms of coordinated service composition. Based on existing theories and building blocks, the design of both the structural as well as the behavioral aspects of the system has been presented and evaluated. The following sections present a summary of this thesis contributions as well as the perspectives that constitute a logical continuation of this work.

Summary of contributions

The diversity of network technologies today ranges from wired-electrical to wireless-optical, and very low bandwidth to very high bandwidth. It is inconceivable that a single protocol algorithm; no matter how long it has been tested and improved can provide good performances in all situations. This fact is now widely accepted by the members of the scientific community whose approach has been to look for solutions that will improve the service provided to applications in very specific contexts. This approach has led to the fast growth of the number of solutions being proposed at the various layers of the protocol stack. The ACT presents a solution to the coordinated instantiation of these mechanisms to be instantiated on end-hosts for end to end communications and does not require the deployment of any new components inside the network.

A generic architecture for multi-layer service composition

Facing the high number of solutions which are proposed at each layer of the communication stack, it is hard to determine the environments for which these improvements are adequate. Indeed, although a new mechanism is generally tested against all previous candidates to fixing a given limitation, the test is only performed in an isolated environment with a relatively small set of varying parameters.

Releasing these new protocols to the wild is a complex task which comes with high responsibilities. In the present state of things, there is no way back. Should there be an error in the description of the contexts a given mechanism is suitable for, the users using the protocol in these environments will get low performance. The ACT presents an architecture which combines the autonomic computing architecture with artificial intelligence decision and learning techniques for solving these limitations.

The definition of the ACT presenting both its structure and the algorithms that govern its behavior have been presented in chapter 2. The presentation that was made aimed at remaining at a high level of abstraction to maintain a generic approach applicable to many concrete cases. The use of autonomic computing techniques which decompose the problem into various layers has been illustrated. Artificial intelligence as well as learning theories have been used to build the decision elements of the system.

Decision and Learning Algorithms for service composition

The software architecture provided by the ACT to support a multi-layer coordinated adaptation allows for all the necessary data for the decision making to be made available. The availability of this information makes the implementation of adaptation algorithms possible.

In order to overcome the limitation of existing systems both in terms of high complexity and lack of extensibility, the approach presented in this manuscript relies on theories developed in the field of artificial intelligence. Indeed, artificial intelligence addresses the problem of having a software component dynamically discover what actions are best to achieve his goals.

To be applied to the context of networking and service composition, the Markov Decision Process formalism has been extended. This extension is a major research contribution of the thesis. The main feature that was added consisted in distinguishing the service provided to an application when using a given service composition from the environment on top of which the composition is executed. However, these two elements are strongly related and constitute the knowledge required to take decisions.

Furthermore, the construction of the model by the learning algorithm does not imply any stationarity properties on the environment. The model is always refined by new experience which allows for the system to be extensible. Indeed, new services can dynamically be integrated into the system

during its execution.

Combining both architectural and behavioral approaches to adaptation

The concepts exposed in chapter 2 for the design of the ACT have been instantiated and extended in a concrete implementation of the ACT. Indeed, the existence of dynamic composition frameworks has raised the need for an additional decision layer in the architecture. Chapter 3 details the implementation of this extension to the ACT to take the services provided by the Enhanced Transport Protocol (ETP) into account. For doing this, the definition of an additional decision layer following the autonomic computing paradigm has lead to the definition of the Autonomic Transport Protocol (ATP). Not only is the ATP capable of performing architectural adaptation by dynamically adapting its composition but it also supports behavioral adaptation through the introduction of Autonomic Micro-Protocols. These Autonomic Micro-Protocols rely on control models which are specific to the mechanisms they target, an example implementation based on automated control theory illustrates their use at the transport layer.

A prototype implementation of this architecture has been tested both through simulations and in emulated network environments. The results have shown that the architecture and algorithms are capable of building a model of the environment in which each possible composition's effect on performance is memorized. This model has shown to be suitable for use by an agent aiming to maintain a given level of performance when the environment in which it evolves changes.

Perspectives

The results presented in chapter 4 show that the ACT is able to provide applications with communication channels which offer them with the most adequate services based on these application's requirements. However, several aspects of the system can be further investigated. The following sections detail these further explorations.

Knowledge Sharing

In the ACT, each machine builds its own model of the environment. Indeed, depending on the history of executions, two instances of the ACT might have information about two disjoint, yet complementary regions of the environment state space.

To reduce the time and number of iterations in the learning phase, the approach that would consist in dynamically obtaining an initial knowledge for the contexts which an agent has not explored yet constitutes an interesting challenge. There are many aspects to this problem which would ease the extension of the approach to new environments. For example, the information gathered by one agent might not be useful to the other if it is unable to instantiate the services that were used to build the knowledge. Additionally, trust and security issues are to be considered not to compromise the system's stability by the passing of false information.

Peer-to-peer networks have recently been proposed to provide connection management services for novel communication systems. The integration of these solutions in the ACT could be easily performed at the Machine Manager level. Indeed, when the information regarding the effects of a given composition are unknown to the ASM, the Machine Manager could try to solve the problem at a higher level by querying peers on the peer-to-peer network for this information.

Services Discovery

The knowledge sharing problem raises the related issue of Services Discovery. In this context, when an ACT instance asks its peers for the best service composition to be used in a given context, the reply it receives might contain services which it is unaware of. For the system to be able to efficiently use information sharing and group knowledge, it is important that every adapting peer has a way of discovering whether or not the services that are contained in a given composition are available to it.

In the context of dynamically composable protocol frameworks, the challenge is interesting as the Micro-Protocols that compose these frameworks are relatively small in size and could be easily dynamically retrieved and instantiated from a central repository or a peer-to-peer like system.

Inclusion of new adaptation levels in the architecture

The architectural and behavioral adaptation models and algorithms of the ACT are used to determine the best service composition for given application goals. These models consider adaptation at the Communication Channel level. However, the ACT architecture identifies other adaptation levels such as the Application and Machine levels.

The extension of the models used to perform adaptation and learning at the Communication Channel level for them to permit a higher level of coordination is a challenging perspective. Moreover, beyond the Machine level, the collaborative adaptation of a group of peers to improve the service provided to some participants is an interesting issue. The field of QoS for peer-to-peer overlay networks constitutes a use case scenario in which such extensions might be applied.

System performances

Finally, the current implementation of the ACT has shown to be quite efficient in terms of speed and computational overhead. However, the tests performed in this work have been performed on a relatively small number of services in a predefined set of network environments.

The static determination of the quantization step for the construction of the environment state as well as the service characterization state spaces might be optimized. Indeed, when the granularity is too high, adjacent environment states lead to the same service characterization with very high probabilities. On the other hand, when granularity is too low, the same network environment leads to many different service characterizations with similar probabilities, making it hard for the decision algorithm to determine a best candidate.

Based on this observation, improvements to the learning algorithms could be introduced for the granularity of the quantization that is performed during learning to be dynamically adapted during the system's execution. Indeed, the granularity does not need to be the same throughout the whole spectrum, several zones of high definition could be defined if required.

Another approach to improving the service performance resides in the enrichment of the decision algorithms. Indeed, the consideration of semantic elements describing the services that can be composed might help eliminate potential candidates from the exploration phase.

LIST OF FIGURES

APPENDIX

Complements to section 4.4.2

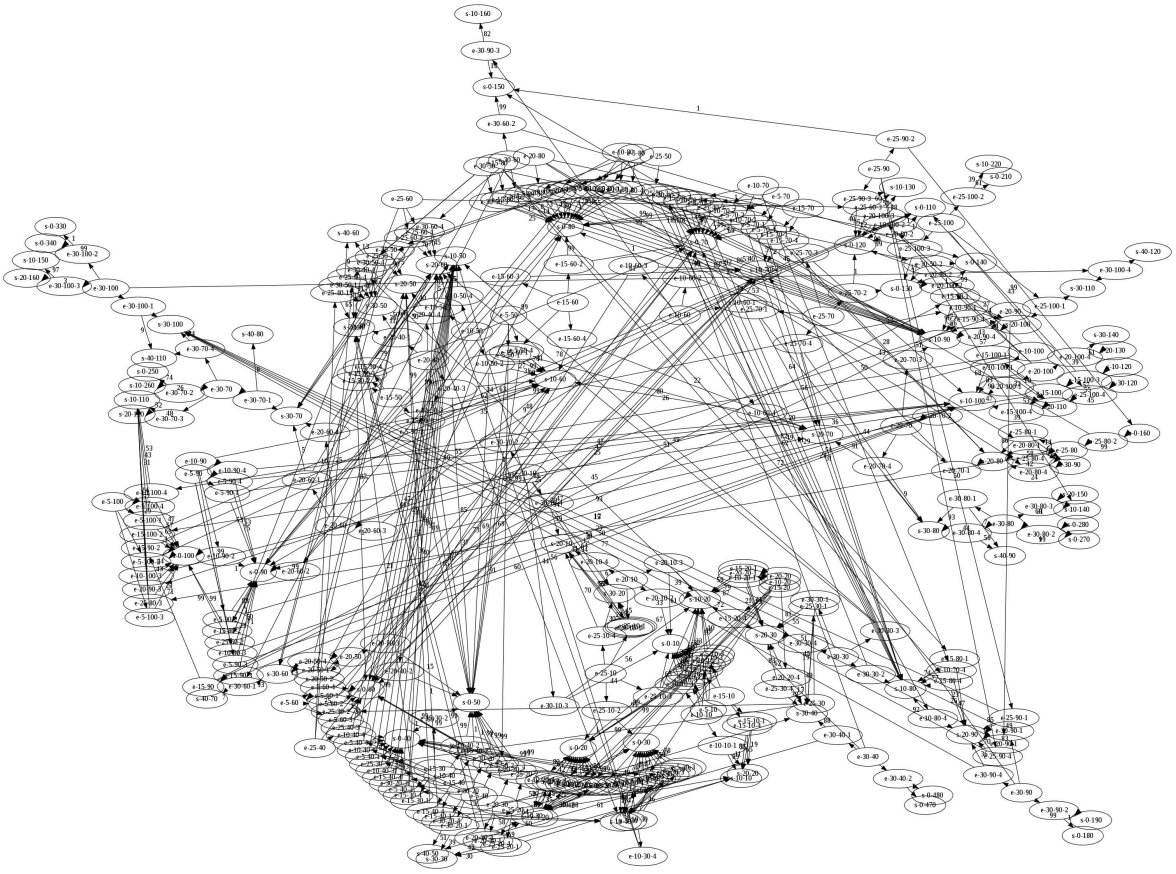


Figure 4.20: Full representation of the MDP obtained through learning

REFERENCES

Referenced Publications

Protocol Composition Frameworks

[F-1] O. B Akan and I. F. Akyildiz. ATL: An adaptive transport layer suite for next-generation wireless internet. *IEEE Journal on Selected Areas in Communications*, Vol. 22(No. 5):802–817, June 2004.

[F-2] P. Amer, C. Chassot, C. Connolly, P. Conrad, and M. Diaz. Partial order transport service for multimedia and other applications. *IEEE ACM Transactions on Networking*, vol.2, no. 5, October 1994.

[F-3] Benjamin Atkin and Kenneth P. Birman. Evaluation of an adaptive transport protocol. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003)*, April 2003.

[F-4] Nina T. Bhatti, Matti A. Hiltunen, Richard D. Schlichting, and Wanda Chiu. Coyote: a system for constructing fine-grain configurable communication services. *ACM Transactions on Computer Systems*, 16(4):321–366, 1998.

[F-5] N.T. Bhatti and R.D Schlichting. Configurable communication protocols for mobile computing. In *The Fourth International Symposium on Autonomous Decentralized Systems*, pages 220–227, 1999.

[F-6] Ken Birman, Robert Constable, Mark Hayden, Christopher Kreitz, Ohad Rodeh, Robbert van Renesse, and Werner Vogels. The Horus and Ensemble Projects: Accomplishments and limitations. In *Proc. of the DARPA Information Survivability Conference & Exposition (DISCEX '00)*, 2000.

[F-7] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. *SIGCOMM Comput. Commun. Rev.*, 20(4):200–208, 1990.

[F-8] Ernesto J. Exposito G. *Design and implementation of a QoS oriented transport protocol for multimedia applications*. PhD thesis, Institut National Polytechnique de Toulouse, Networks and Telecommunications, 2003.

[F-9] N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, 1991.

[F-10] E. Kohler, M. Handley, and S. Floyd. Rfc4340: Datagram congestion control protocol (dccp). Technical report, IETF, March 2006.

[F-11] Sergio Mena, Xavier Cuvellier, Christophe Gregoire, and Andre Schiper. Appia vs. cactus: Comparing protocol composition frameworks. In *Proceeidngs of the 22nd Symposium on Reliable Distributed Systems (2003)*, 2003.

[F-12] H. Miranda, A. Pinto, and L. Rodrigues. Appia: A flexible protocol kernel supporting multiple coordinated channels. In *Proc. 21st International conference on Distributed Computing Systems (ICDCS-21), (Phoenix, Arizona, USA)*, pages 707–710, 2001.

[F-13] Dennis M. Ritchie. A stream input-output system. *AT&T Bell Laboratories Technical Journal*, 63(8):1897–1910, 1984.

[F-14] D. C. Schmidt, D. F. Box, and T. Suda. ADAPTIVE — A Dynamically Assembled Protocol Transformation, Integration and eValuation Environment. *Journal of Concurrency: Practice and Experience*, 5(4):269–286, 1993.

[F-15] Douglas C. Schmidt, Donald F. Box, and Tatsuya Suda. ADAPTIVE: A flexible and adaptive transport system architecture to support lightweight protocols for multimedia applications on high-speed networks. In *Proceedings of the 1st Symposium on High-Performance Distributed Computing (HPDC-1), (Syracuse, New York)*, pages 174–186, 1992.

[F-16] R. Stewart. Rfc4960: Stream control transmission protocol. Technical report, IETF, September 2007.

[F-17] Gary T. Wong, Matti A. Hiltunen, and Richard D. Schlichting. A configurable and extensible transport protocol. In *Proceedings of INFOCOM*, pages 319–328, Anchorage, Alaska, USA, April 2001.

[F-18] R. Wu, A. Chien, M. Hiltunen, R. Schlichting, and S. Sen. A high performance configurable transport protocol for grid computing. In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005), Cardiff, Wales*, 2005.

Protocol Mechanisms and Tools

[M-1] Bernard Berthomieu, Florent Peres, and Francois Vernadat. *Modeling and Verification of Real-Time Systems*, chapter Time Petri Nets: Analysis Methods and Verification with TINA, pages 19–48. Wiley, 2008.

[M-2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC2475: An architecture for differentiated service. Technical report, IETF, December 1998.

[M-3] R. Braden, D. Clark, and S. Shenker. RFC1633: Integrated services in the internet architecture: an overview. Technical report, IETF, 1994.

[M-4] Christof Brandauer and Thomas Fichtel. MINER - a measurement infrastructure for network research. In *Proceedings of the 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM)*, April 2009.

[M-5] M. Callejo-rodriguez and J. Enriquez-Gabeiras. Bridging the standardization gap to provide qos in current ngn architectures. *Communications Magazine, IEEE*, 46(10):132–137, October 2008.

[M-6] S. Cen, P. Cosman, and G. Voelker. End-to-end differentiation of congestion and wireless losses. In *ACM Multimedia Computing and Networking*, 2003.

[M-7] M. Chen and A. Zakhor. Rate control for streaming video over wireless. *IEEE Wireless Communications*, Aug.:32+, 2005.

[M-8] V. Fineberg. A practical architecture for implementing end-to-end qos in an ip network. *Communications Magazine, IEEE*, 40(1):122–130, Jan 2002.

[M-9] Sally Floyd. RFC2914: Congestion control principles.

[M-10] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[M-11] M. Gineste, N. Van Wambeke, E. Expósito, C. Chassot, and L. Dairaine. A cross-layer approach to enhance QoS for multimedia applications over satellite. *Wireless Personal Communications*, 2008.

[M-12] Mathieu Goutelle, Yunhong Gu, Eric He (editor), Sanjay Hegde, Rajkumar Kettimuthu, Jason Leigh, Pascale Vicat-Blanc/Primet, Michael Welz, and Chaoyue Xiong. A survey of transport protocols other than standard tcp. In *Global Grid Forum Data Transport Research Group*. February 2004.

[M-13] M. Handley, J. Pahdye, S. Floyd, , and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol specification. Internet draft draft-ietf-tsvwg-tfrc02. txt, work in progress, January 2003.

[M-14] S. Hemminger. Network emulation with netem. In *Linux Conf Au*, April 2005.

[M-15] Zhu Hua and I. Chlamtac. Performance analysis for ieee 802.11e edcf service differentiation. *IEEE Transactions on Wireless Communications*, 8:1779–1788, 2005.

[M-16] Sami Iren, Paul D. Amer, and Phillip T. Conrad. The transport layer: tutorial and survey. *ACM Computing Surveys*, 31(4):360–404, 1999.

[M-17] S. Khan, K.F. Li, and E.G. Manning. Padma: an architecture for adaptive multimedia systems. volume 1, pages 105–108 vol.1, Aug 1997.

[M-18] Emmanuel Lochin and Laurent Dairaine ANDGuillaume Jourjon. gtfrc, a tcp friendly qos-aware rate control for diffserv assured service. *Springer Telecommunication Systems*, 10.1007/s11235-006-9004-2, ISSN: 1018-4864 (Print) 1572-9451 (Online), September 2006 2006.

[M-19] K. Nichols, S. Blake, F. Baker, and D. Black. RFC 2474 : Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers, 1998.

[M-20] N. Samaan and A. Karmouch. An automated policy-based management framework for differentiated communication systems. *Selected Areas in Communications, IEEE Journal on*, 23(12):2236–2247, Dec. 2005.

[M-21] G. T'Hooft. Timer description in ccs (milner), lotos (iso) and timed lotos (quemada-fernandez). a case analysis. *SIGCOMM Comput. Commun. Rev.*, 18(1-2):31–62, 1988.

[M-22] Guang Yang, Ling jyh Chen, Tony Sun, Mario Gerla, and M. Y. Sanadidi. Real-time streaming over wireless links: A comparative study. In *In Proceedings of the IEEE ISCC. IEEE*, 2005.

Autonomic Computing and Artificial Intelligence

[AAI-1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2007.

[AAI-2] S. Balasubramaniam and J. Indulska. Vertical handover supporting pervasive computing. *Elsevier Computer Communication Journal, Special Issue on 4G/Future Wireless networks*, vol. 27/8:708–719, 2004.

[AAI-3] Philippe Boinot, Renaud Marlet, Jacques Noye, Gilles Muller, and Charles Consel. A declarative approach for designing and developing adaptive components. In *Automated Software Engineering*, pages 111–, 2000.

[AAI-4] I. Bouassida Rodriguez, K. Drira, C. Chassot, and M. Jmaiel. Context-aware adaptation approach for group communication support applications with dynamic architecture. *System and Information Sciences Notes (SISN) Journal*, 2, 2007.

[AAI-5] P. G. Bridges, W-K. Chen, M. A. Hiltunen, and Richard D. Schlichting. Supporting coordinated adaptation in networked systems. In *8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), Elmau, Germany*, May 2001.

[AAI-6] Richard Dearden, Nir Friedman, and David Andre. Model based bayesian exploration. In *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 150–159, 1999.

[AAI-7] Arkady Epshteyn, Adam Vogel, and Gerald DeJong. Active reinforcement learning. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 296–303, New York, NY, USA, 2008. ACM.

[AAI-8] Ernesto Exposito, Mathieu Gineste, Laurent Dairaine, and Christophe Chassot. Building self-optimized communication systems based on applicative cross-layer information. *Computer Standards & Interfaces*, 31(2):354 – 361, 2009.

[AAI-9] Ernesto Expósito, Nicolas Van Wambeke, Christophe Chassot, and Khalil Drira. Introducing a cross-layer interpreter for multimedia streams. *Elsevier Computer Networks*, 52:1125–1141, 2008.

[AAI-10] K. Farkas, O. Wellnitz, M. Dick, X. Gu, M. Busse, W. Effelsberg, Y. Rebahi, D. Sisalem, D. Grigoras, K. Stefanidis, and D.N. Serpanos. Real-time service provisioning for mobile and wireless networks. *Elsevier Computer Communication Journal*, Vol. 29, No. 5:540–550, march 2006.

[AAI-11] Robert Givan, Sonia Leach, and Thomas Dean. Bounded parameter markov decision processes. *Artificial Intelligence*, 122:234–246, 1997.

[AAI-12] Xiaoyuan Gu, Xiaoming Fu, Hannes Tschofenig, and Lars C. Wolf. Towards self-optimizing protocol stack for autonomic communication: Initial experience. In *WAC*, pages 186–201, 2005.

[AAI-13] Xiaoyuan Gu, J. Strassner, Jiang Xie, L.C. Wolf, and T. Suda. Autonomic multimedia communications: Where are we now? *Proceedings of the IEEE*, 96(1):143–154, Jan. 2008.

[AAI-14] K. Guennoun. *Dynamic architectures in the context of component-based and service-oriented Applications*. PhD thesis, Paul Sabatier University, Toulouse, France, 2006.

[AAI-15] Karim Guennoun, Khalil Drira, Nicolas Van Wambeke, Christophe Chassot, François Armando, and Ernesto Exposito. A framework of models for qos-oriented adaptive deployment of multi-layer communication services in group cooperative activities. *Computer Communications*, 31(13):3003–3017, 2008.

[AAI-16] John Hertz, Richard G. Palmer, and Anders S. Krogh. *Introduction to the Theory of Neural Computation*. Perseus Publishing, 1991.

[AAI-17] IBM. An architectural blueprint for autonomic computing. *White Paper*, 2006.

[AAI-18] Christophe Jelger, Christian Tschudin, Stefan Schmid, and Guy Leduc. Basic abstractions for an autonomic network architecture. *Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007*, pages 1–6, 18-21 June 2007.

[AAI-19] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[AAI-20] R. Landry, K. Grace, and A. Saidi. On the design and management of heterogeneous networks : A predictability-based perspective. *IEEE Communications Magazine*, Volume 42, Issue 11:Page(s):80 – 87, November 2004.

[AAI-21] Jose Mocito, Liliana Rosa, Nuno Almeida, Hugo Miranda, Luis Rodrigues, and Antonia Lopes. Context adaptation of the communication stack. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)*, Colombus, OH, USA, June 2005. Colombus.

[AAI-22] Manish Parashar and Salim Hariri. Autonomic computing: An overview. In *Unconventional Programming Paradigms*, volume 3566 of *LNCS*, pages 257–269. Springer, 2005.

[AAI-23] Ra Penn and Inman Harvey. Learning from delayed rewards. In *C. Watkins, King's College*, pages 352–357. MIT Press, 2004.

[AAI-24] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, April 1994.

[AAI-25] Hariharan S. Rahul, Hari Balakrishnan, and Srinivasan Seshan. An end-system architecture for unified congestion management. In *In Proceedings of 7th Workshop on Hot Topics in Operating Systems*, pages 52–57, 1999.

[AAI-26] Liliana Rosa, Antónia Lopes, and Luís Rodrigues. Policy-driven adaptation of protocol stacks. In *Proceedings of the IEEE Self-adaptability and self-management of context-aware systems workshop (SELF)*, page 5, Santa Clara (CA), USA, July 2006.

[AAI-27] Jin Xiao and R. Boutaba. Qos-aware service composition and adaptation in autonomic communication. *IEEE Journal on Selected Areas in Communications*, 23(12):16, Dec 2005.

Author Publications

Book Chapters

[1] M. Diaz, J. Enrique-Gabeiras, L. Baresse, A. Beben, W. Burakowski, M.A Callejo Rodriguez, J. Carapinha, O. Dugeon, E. Exposito, M. Gineste, E. Mingozzi, E. Monteiro, A. Pietrabissa, S.F. Racaru, J. Sliwinski, G. Stea, H. Tarasiuk, N. Van Wambeke, and M. Wulff. *End-to-End Quality of Service over Heterogeneous Networks*, chapter Enhanced Transport Protocols, pages 131–179. 2008.

[2] N. Van Wambeke, F. Armando, C. Chassot, K. Guennoun, K. Drira, and E. Exposito. *Wireless and Mobile Networking*, chapter Towards the use of models for autonomic network management. Springer, 2008.

[3] Nicolas Van Wambeke, Ernesto Expósito, Guillaume Jourjon, and Emmanuel Lochin. *End-to-End Quality of Service over Heterogeneous Networks*, chapter Enhanced Transport Protocols, pages 111–129. 2008.

International Peer Reviewed Journal Articles

[1] Ismael Bouassida Rodriguez, Nicolas Van Wambeke, Khalil Drira, Christophe Chassot, and Mohamed Jmaiel. Multi-layer coordinated adaptation based on graph refinement for cooperative activities. *Communications of SIWN*, 2008.

[2] Ernesto Expósito, Nicolas Van Wambeke, Christophe Chassot, and Khalil Drira. Introducing a cross-layer interpreter for multimedia streams. *Elsevier Computer Networks*, 52:1125–1141, 2008.

[3] M. Gineste, N. Van Wambeke, E. Expósito, C. Chassot, and L. Dairaine. A cross-layer approach to enhance QoS for multimedia applications over satellite. *Wireless Personal Communications*, 2008.

[4] Karim Guennoun, Khalil Drira, Nicolas Van Wambeke, Christophe Chassot, Francois Armando, and Ernesto Expósito. A framework of models for QoS-oriented adaptive deployment of multi-layer communication services in group cooperative activities. *Elsevier Computer Communication*, 2008.

[5] N. Van Wambeke, F. Armando, A. Abdelkefi, C. Chassot, K. Guennoun, and K. Drira. Models and architectures for autonomic networking. *International Journal of Buisiness Data Communications and Networking*, 5(3), 2009.

[6] Nicolas Van Wambeke, Francois Armando, Christophe Chassot, and Ernesto Expósito. A model-based approach for self-adaptive transport protocols. *Elsevier Computer Communication*, 2008. Special Issue on End-to-end Support over Heterogeneous Wired-Wireless Networks.

International Peer Reviewed Conference Articles

[1] Pedro Andres Aranda Gutierrez, Adam Flizikowski, Nicolas Van Wambeke, Francois Armando, Christophe Chassot, Christof Niephaus, David Wagner, Ilka Miloucheva, and Simon Pietro Romano. Netqos policy management architecture for flexible qos provisioning in future internet. In *Proceedings of the Second International Conference and Exhibition on NEXT GENERATION MOBILE APPLICATIONS, SERVICES and TECHNOLOGIES (NGMAST'08)*, Cardiff, Wales, UK, September 16-19 2008.

[2] Francois Armando, Nicolas Van Wambeke, and Christophe Chassot. Introducing a collaborative congestion control based on tfrc. In *Proceedings of the World Congress on Engineering and Computer Science (WCECS)*, San Francisco, USA, October 2008.

[3] Francois Armando, Nicolas Van Wambeke, Christophe Chassot, Ernesto Expósito, and Khalil Drira. A framework for the dynamic configuration of adaptive transport protocols. In *Proceedings of the International Conference on Wireless Information Networks and Systems (WINSYS 2007)*, pages 275–283, Barcelona (Spain), 28-31 July 2007.

[4] Ernesto Expósito and Nicolas Van Wambeke. Design principles for an autonomic transport protocol framework. In *Proceedings of the Latin American Autonomic Computing Symposium (LAACS)*, pages 19–26, Gramado, Brazil, September 2008.

[5] Ernesto Expósito, Nicolas Van Wambeke, Christophe Chassot, and Michel Diaz. Uml 2.0 based methodology for designing and developing real-time and qos-oriented communication protocols. In *Proceedings of the ANIPLA International Congress 50th Anniversary 1956-2006, Methodologies for emerging technologies in automation*, 2006.

[6] Sathya Rao, Sophia Khavtasi, Christophe Chassot, Nicolas Van Wambeke, Francois Armando, Simon Pietro Romano, and Tobia Castaldi. Qos management in the future internet. In *Proceeding of the International Conference on Information and Communication Technologies (ICT08)*, Paris, France, July 04-06 2008.

[7] Nicolas Van Wambeke, Francois Armando, Christophe Chassot, and Ernesto Expósito. Architecture and models for self-adaptability of transport protocols. In *Proceedings of the 21st IEEE International Conference on Advanced Information Networking and Applications (AINA-07) Workshops*, Niagara Falls, Canada, May 21-23 2007.

[8] Nicolas Van Wambeke, Francois Armando, Christophe Chassot, and Ernesto Expósito. A model based approach to communication protocol's self-adaptation. In *Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE 2007)*, Seoul, South Korea, April 26th-28th 2007.

[9] Nicolas Van Wambeke, Francois Armando, Karim Guennoun, Khalil Drira, Ernesto Expósito, and Christophe Chassot. Towards the use of models for autonomic network management. In *Proceedings of the 13th IFIP International Conference on Personal Wireless Communications (PWC'2008)*, Toulouse, France, October 1-3 2008.

[10] Nicolas Van Wambeke, Ernesto Expósito, and Michel Diaz. Transport layer qos protocols: the micro-protocol approach. In *Proceedings of the 1st OpenNet Workshop*, Diegem, Belgium, 27-29 March 2007. Service Quality and IP Network Business: Filling the Gap.

National Peer Reviewed Conference Articles

[1] Nicolas Van Wambeke, Francois Armando, Christophe Chassot, and Ernesto Expósito. Un modele de decision pour l'auto-adaptation des protocoles de communication. In *Proceedings of the Colloque Francophone sur l'Ingenierie des Protocoles (CFIP)*, Les Arcs, 25-28 Mas 2008.

[2] Nicolas Van Wambeke and FranÃ§ois Armando. De la faisabilite d'un controle de congestion collaboratif. In *Actes de la 6eme MAnifestation des Jeunes Chercheurs en STIC (MajecSTIC)*, 2009.

European Project Reports

NetQoS

[1] C. Chassot, E. Exposito, N. Van Wambeke, F. Armando, I. Bouassida, K. Drira, C. Brandauer, F. Strohmeier, P. Dorfinger, P. A. Aranda Gutierrez, A. Flizikowski, K. Samp, I. Miloucheva, S. Avallone, L. Vollero, S. Pietro Romano, M. Michalak, M. Roth, and S. Rao. D2.1 - QoS solution analysis and user/application requirements and business scenarios. Technical report, IST FP6 - NetQoS, 2007.

[2] C. Chassot, E. Exposito, N. Van Wambeke, F. Armando, C. Brandauer, T. Fichtel, S. Kavthasi, S. Rao, I. Miloucheva, D. Wagner, A. Flizikowski, R. Marcinczak, S. Avallone, S. Pietro Romano, and Alberto Dainotti. D2.3 - policy specification. Technical report, IST FP6 - NetQoS, 2007.

[3] C. Chassot, E. Exposito, N. Van Wambeke, F. Armando, C. Brandauer, T. Fichtel, S. Kavthasi, S. Rao, I. Miloucheva, D. Wagner, A. Flizikowski, R. Marcinczak, S. Avallone, S. Pietro Romano, and Alberto Dainotti. D2.4 - specification of policy-oriented measurement, learning, and context identification. Technical report, IST FP6 - NetQoS, 2007.

[4] C. Chassot, E. Exposito, N. Van Wambeke, F. Armando, C. Brandauer, S. Khavatsi, S. Rao, S. Avallone, S.P. Romano, A. Pescape, A. Botta, A. Dainotti, W. De Donato, L. Vollero, and P. Aranda Gutierrez. D2.6: Implementation of policy-oriented measurement, learning, and context identification. Technical report, IST FP6 - NetQoS, 2008.

[5] C. Chassot, E. Exposito, N. Van Wambeke, F. Armando, C. Brandauer, J.B. Neuhaus, M. Roth, S. Rao, I. Miloucheva, A. Flizikowski, R. Marcinczak, P. A. Aranda Gutierrez, S. Avallone, and S. Pietro Romano. D2.2 - design of the NetQoS policy architecture. Technical report, IST FP6 - NetQoS, 2007.

[6] A. Flizikowski, R. Marcinczac, I. Miloucheva, S. Khavatsi, S. Rao, S. Avallone, S.P. Romano, N. Van Wambeke, F. Armando, E. Exposito, and C. Chassot. D2.5: Policy implementation. Technical report, IST FP6 - NetQoS, 2008.

[7] P. A. Aranda Gutierrez, C. Chassot, E. Exposito, F. Armando, N. Armando, N. Van Wambeke, C. Brandauer, S. Khavtasi, S. Rao, D. Wagner, and A. Flizikowski. D3.1 - specification of policy-oriented measurement, learning, and context identification. Technical report, IST FP6 - NetQoS, 2007.

[8] Pedro A. Gutierrez, Fillip Trivunac, Ilka Miloucheva, David Wagner, Christian Niephaus, Christophe Chassot, Nicolas Van Wambeke, Christof Brandauer, Adam Flizikowski, François Armando, Sathya Rao, and Simon Pietro Romano. D3.4: Performance assessment. Technical report, IST FP6 - NetQoS, 2008.

[9] Pedro A. Aranda Gutierrez, Sathya Rao, Sophia Khavtasi, Christof Brandauer, Adam Flizikowski, Christophe Chassot, Nicolas Van Wambeke, François Armando, Ilka Miloucheva, David Wagner, and Simon Pietro Romano. D3.2: System integration (first prototype) and test report. Technical report, IST FP6 - NetQoS, 2008.

[10] Ilka Miloucheva, David Wagner, Christian Niephaus, Christophe Chassot, Nicolas Van Wambeke, Pedro A. Gutierrez, Fillip Trivunac, Christof Brandauer, Adam Flizikowski, Sophia Khavtasi, François Armando, Sathya Rao, and Simon Pietro Romano. D3.3: Trials execution. Technical report, IST FP6 - NetQoS, 2008.

[11] Ilka Miloucheva, David Wagner, Christian Niephaus, Christophe Chassot, Nicolas Van Wambeke, Pedro A. Gutierrez, Fillip Trivunac, Christof Brandauer, Adam Flizikowski, Sophia Khavtasi, François Armando, Sathya Rao, and Simon Pietro Romano. D3.5: Validation and exploitation report. Technical report, IST FP6 - NetQoS, 2008.

EuQoS

[1] All EuQoS Partners. D1.2.2 - euqos architecture update for phase 2. Technical report, IST FP6 EuQoS, 2007.

[2] All EuQoS Partners. D2.2.3 - final validation of the euqos system by simulation. Technical report, IST FP6 - EuQoS, 2007.

[3] All EuQoS Partners. D4.2.1 - integrated euqos system software architecture for application use cases and api for application driving. Technical report, IST FP6 - EuQoS, 2007.

RÉSUMÉ EN FRANÇAIS

Les dernières évolutions de l'Internet se sont traduites par l'émergence de nouvelles applications distribuées et par la multiplication des technologies réseaux (sans fils, mobiles...) ainsi que des services offerts par les opérateurs sur de nouveaux types de terminaux (portable, PDA...). L'enjeu socio économique majeur de ces avancées est le futur Internet ambiant, à la fois ubiquitaire et intelligent, au travers duquel l'utilisateur pourra, quelle que soit sa localisation et son point d'accès, bénéficier d'une qualité de service (QdS) maximale compatible avec l'environnement applicatif et réseau courant, hétérogène et dynamique. Dans cet horizon, la thèse présente une architecture ainsi que des modèles et algorithmes permettant de réaliser une composition dynamique et auto-adaptative des services fournis par les multiples mécanismes de QdS existants. L'approche proposée repose sur l'adaptabilité dynamique et coordonnée, à la fois du comportement et de l'architecture des protocoles composant la pile de communication. La démarche suivie se base sur la théorie de l'intelligence artificielle et l'apprentissage et propose la spécification, l'implémentation et l'évaluation d'un système de communication adaptatif en fonction à la fois, des exigences applicatives liées aux flux manipulés, et des contraintes de l'environnement de communication. Enfin, l'évaluation faite des modèles de décision et d'apprentissage illustre comment le système permet de répondre à son objectif et valide ainsi les concepts qui sont proposés dans cette thèse.

Mots-clés: Système de communication, Intelligence Artificielle, Adaptabilité, Architecture, Protocole de transport, Apprentissage par renforcement Qualité de Service, Réseaux Informatiques

ENGLISH ABSTRACT

The recent evolutions of the Internet have been materialized by the emergence of new distributed application as well as the rapid increase of the number of network technologies (wireless, mobile...). These advances have allowed new services to be proposed by operators using a new type of terminals (laptop, PDA...). The major socio economic stake that these advances represent is materialized by the future ambient Internet, which is both ubiquitous and intelligent, through which the user will be able, whatever his location or access point, benefit from the best possible Quality of Service (QoS) given its current applicative and network environment. In this context, the thesis presents an architecture as well as models and algorithms that allow for a dynamic self-adaptive composition of services provided by existing QoS mechanisms. The approach is based on the dynamic and coordinated adaptation of both the behaviour and the architecture of the protocols that compose the communication stack. The work finds its inspiration in the theory of artificial intelligence and learning and provides the specification, implementation and evaluation of a self-adaptive communication system taking into account both, the applicative requirements for the transported streams and the constraints of the environment on which communication takes place. Finally, the evaluation of the decision and learning models illustrates how the system allows to fulfil its objective and validates the concepts that are proposed in this thesis.

Keywords: Communication System, Artificial Intelligence, Adaptation, Architecture Transport Protocol, Reinforcement Learning, Quality of Service, Computer Networks