Methodology, models and paradigms for a next generation transport layer design

Ernesto Expósito

Mémoire d'Habilitation à Diriger des Recherches

Préparée au

Laboratoire d'Analyse et d'Architectures des Systèmes du CNRS

En vue de l'` btenti` n du Diplôme d'Habilitati` n à Diriger des Recherches
de l'Institut Nati` nal P` lytechnique de T` ul` use (INPT)

Spécialité Informatique

par

Ernesto EXPOSITO

Maître de c` nférences à l'Institut Nati` nal de Sciences Appliquées (INSA) de T` ul` use
Chercheur au LAAS-CNRS

Titre du mém` ire :

Méthodologie, modèles et paradigmes pour la conception d'une couche transport de nouvelle génération

Methodology, models and paradigms for a next generation transport layer design

Date et jury de soutenance :

9 décembre 2010 à 10h30, salle de conférences du LAAS-CNRS

Isabelle	Guerin-Lass` us	Pr` fesseur à l'Université de Ly` n	Rapp` rteur
Francis	Lepage	Pr` fesseur à l'Université H. P` incare	Rapp` rteur
Flavi`	Oquend`	Pr` fesseur à l'Université de Bretagne Sud	Rapp` rteur
Laurence	Duchien	Pr` fesseur à l'Université de Lille	Examinateur
Christ` phe	Chass` t	Pr` fesseur à l'Université de T` ul` use	Examinateur
Khalil	Drira	Directeur de recherche au CNRS	Examinateur
Michel	Diaz	Directeur de recherche au CNRS	C` rresp` ndant

Décembre 2010

Guide de lecture

Ce mémoire (écrit en anglais) présente les travaux de recherche que j'ai menés au LAAS-CNRS depuis 1999 et au laboratoire National ICT Australia lors de mon séjour de post-doc en 2004 et 2005.

Ce mémoire est organisé en trois chapitres :

- Le chapitre 1 présente un état de l'art des protocoles de transport et propose une méthodologie et une implémentation sous la forme d'un modèle sémantique pour la conception de l'architecture de la couche transport de prochaine génération.
- Le chapitre 2 présente mes travaux relatifs à la conception (UML) et à l'implémentation (JAVA) d'un protocole de transport orienté composants. Il en propose, en perspective, une extension suivant les paradigmes des architectures orientées services et basées composants.
- Le chapitre 3 présente mes contributions aux stratégies d'adaptabilité guidées par les modèles pour gérer l'adaptation comportementale et structurelle des protocoles de transport. En perspective, la conception et le développement des extensions à donner aux protocoles de transport orientés composants pour aboutir à des propriétés d'adaptabilité puis d'autonomie (suivant le modèle de l'*autonomic computing*) sont présentés.

Finalement :

- Les conclusions ainsi que la synthèse des perspectives de recherche dégagées dans les chapitres précédents sont présentées.
- Une annexe détaille les implémentations du modèle sémantique présentées dans les chapitres 1 et 2, et la façon dont elles ont été évaluées et validées.

Résumé
Les thèmes de recherche développés dans cette habilitation portent sur les méthodologies guidées par les modèles sémantiques et les paradigmes architecturaux nécessaires pour la conception et le développement d'une couche transport de nouvelle génération.
Une première partie présente un état de l'art des protocoles de transport et introduit une méthodologie guidée par les modèles et une implémentation sous la forme d'un modèle sémantique pour la conception des protocoles de transport avancés.
Une deuxième partie présente nos travaux relatifs à la conception (UML) et l'implémentation (JAVA) d'un protocole de transport orienté composants. Il en propose, en perspective, une extension suivant les paradigmes des architectures orientées services et basées composants.
Une troisième partie présente nos contributions aux stratégies d'adaptabilité guidées par les modèles pour gérer l'adaptation comportementale et structurelle des protocoles de transport.
En perspective, la conception et le développement d'une couche transport orientée composants et orientée services pour aboutir à des propriétés d'adaptabilité puis d'autonomie suivant le cadre de l'autonomic computing, et basés sur les ontologies, sont présentés.

Mots-clés : protocoles de transport, architectures guidées par le modèles, ontologies, architectures basées composants, architectures orientées services, *autonomic computing*

Summary
The research domains developed in this dissertation include a model-driven methodology based on semantic models and various architectural paradigms aimed at designing and developing a transport layer of new generation.
The first part presents the state of the art of transport protocols and introduces a model driven methodology and an ontology semantic model implementation aimed at designing advanced transport protocols.
The second part presents our work related to the design (UML) and the implementation (JAVA) of a component-based transport protocol. An extension to this protocol based on the service-component and the service-oriented architectures is proposed.
A third part presents our contributions in the area of model-driven adaptive strategies aimed at managing behavioral and structural adaptation of transport protocols.
Finally, a perspective is proposed for designing and developing a transport layer based on the component-oriented and the service-oriented approaches and integrating the autonomic computing framework and the semantic dimension provided by ontologies.

Keywords: transport protocols, model-driven architectures, ontologies, component-based architectures, service-oriented architectures, autonomic computing

TABLE DE MATIÈRES

General Introduction

The accelerated development of Internet and the multitude of networked mobile devices (e.g. smart-phones, PDAs, tablets, netbooks and laptops) has facilitated the development of a vast family of distributed multimedia applications (e.g. VoD, visio-conferencing, IPTV, VoIP, etc.).

Requirements and preferences of these applications become very complex when compared to traditional downloading, web-browsing or emailing applications, for which a reliable and ordered transport service (such as the one offered by the traditional TCP protocol) operating over a wired Best-Effort network service is well suited. However, this new generation of distributed multimedia applications present more complex requirements of Quality of Service (QoS), mainly expressed in terms of time (e.g. end-to-end delay, multimedia synchronization, jitter), bandwidth (e.g. high and variable bandwidth) and reliability (e.g. tolerance for partial reliability and partial order) requirements.

In the past years, several initiatives have been carried out to enhance the basic Best-Effort network service in order to provide new QoS-oriented service models (e.g. DiffServ, IntServ, etc). Moreover, new technologies providing high speed, wireless and mobile network services have deeply modified the QoS characterization of the network layer thus leading to a more complex service model in terms of bandwidth, losses, delay or jitter. Furthermore, networked applications running on mobile devices are exposed to an even more complex service model due to the dynamicity of perceived QoS when moving from high-speed and high-bandwidth networks (e.g. ADSL networks at home), to variable bandwidth and high delay networks (e.g. when operating over WiFi or 3G mobile wireless networks).

This important evolution of application and network layers has deeply impacted the traditional transport layer. Indeed, traditional transport protocols (i.e. TCP and UDP) were well dimensioned to the original best-effort network model. However, specializations of transport mechanisms were required to cope with new network technologies (e.g. TCP extensions for satellite or Wi-Fi networks). Likewise, new protocols such as DCCP, SCTP and MPTCP have been proposed to enhance traditional protocols in order to provide new specialized transport functions (e.g. more adapted network congestion avoidance strategies, multi-homing support for mobility, or multipath support for devices integrating multiple network interfaces). Figure 1 summarizes the application, transport, network and physical layers evolution and illustrates the complexity involved in providing the adequate adaptation service at the transport layer.

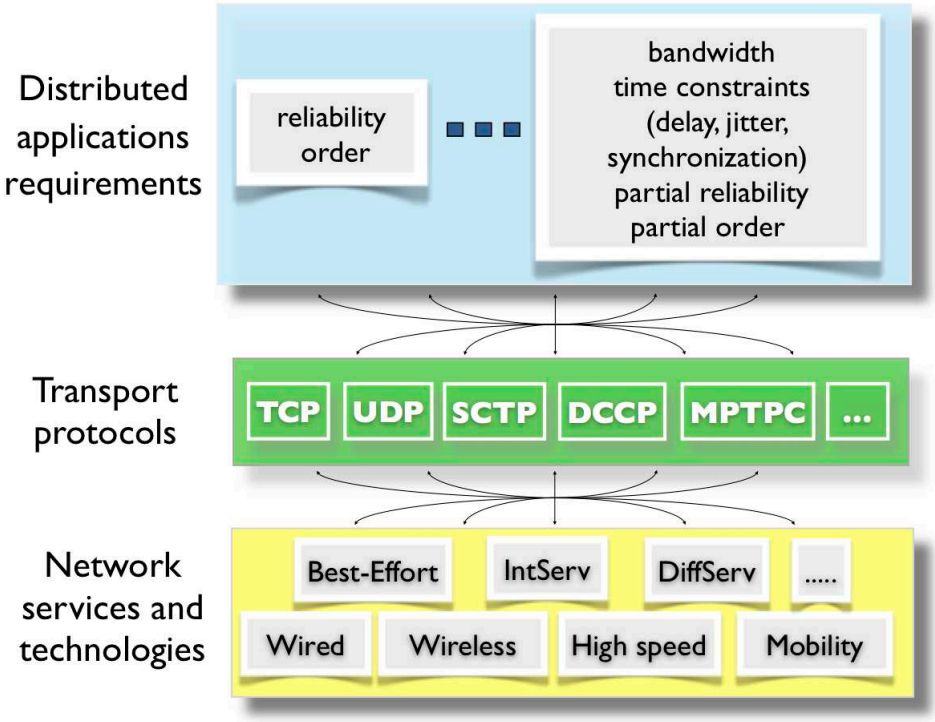

Figure 1. Problem context

Even if the transport layer evolution represents a classical example of software change, no much effort has been invested in applying advanced software engineering practices aimed at preparing the basis for the new extensions and specializations that will be certainly required in the future.

We anticipate that an adequate model-driven engineering methodology would facilitate the design of a flexible architecture providing the required extensibility and reusability capabilities in order to incorporate future protocol extensions and specializations, also adapted to the diversity of applications, network services and user devices.

Moreover, new software architecture paradigms enabling services characterization and dynamic service discovery, selection, composition and deployment should be formally integrated in the transport layer architecture.

Furthermore, an adequate framework enabling the design of self-adapting transport services in dynamic and heterogenous network environments, in order to satisfy a large diversity of applications requirements, should also be incorporated within the architecture of the next generation transport layer.

This dissertation presents our work in defining new protocols and in applying a specialized software engineering methodology to implement the corresponding services. This methodology based on semantic models and integrating service-oriented, component-based and autonomic computing frameworks is intended to design a next generation transport layer.

1. Summary of our contributions

This dissertation presents our research work in transport protocols and proposes a methodology and a set of elementary frameworks and paradigms intended to guide the design and the development of the next generation transport layer:

- The methodology proposed is based on a model-driven architecture design, initiated by abstract standard models of the transport layer and guided by the trends and the evolutions of transport protocols specifications and implementations. The resulting model is implemented by an ontology incorporating the semantic related to the services, protocols, functions and mechanisms of the transport layer.
- Component-based and service-oriented architecture approaches represent a first paradigm guiding the transport layer design. Based on the semantic model of the transport layer, a service-oriented architecture able to dynamically offer the most adequate transport service based on the application requirement expressions and on the available transport protocols and network services has been designed. Likewise, a component-based transport layer architecture enabling the dynamic composition of reusable transport mechanisms based on the semantic associated to the expected final service has been developed.
- The second paradigm represented by the autonomic computing approach is related with the fundamental adaptation function that needs to be guaranteed by the transport layer. While the first paradigm focuses on the adequate selection and composition of transport protocols and mechanisms, the autonomic computing approach is aimed at providing a framework to design and develop the adequate adaptation functionalities to be provided by the transport protocols during the data transfer phase in order to cope with the dynamicity of the service offered by the network layer.

Finally, the conclusions and perspectives of this work are presented. Guidelines and future works aimed at designing and developing an ontology-driven, component-based, service-oriented and autonomic computing architectural framework intended to be integrated within the next generation transport layer are presented. Lessons learned from the applied methodology and from our work on component-based and adaptive transport protocols as well as several perspective studies to be carried out in order to design and develop self-managing (i.e. discovery, selection, composition, deployment and adaptation) autonomic properties of the next generation of transport protocols are also presented.

2. Structure of the dissertation

Figure 2 presents the structure of the dissertation and summarizes our research work in the area of transport protocols as well as the methodology, the frameworks and the paradigms proposed.

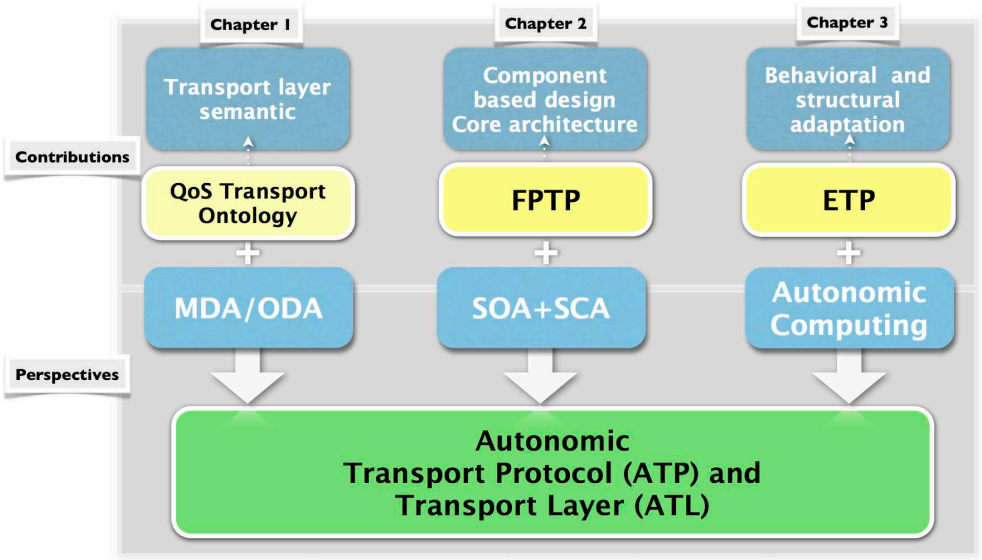

Figure 2. Structure of the dissertation, contributions and perspectives

This study is structured as following:

- Chapter 1 presents the state of the art on transport services and protocols and proposes a first contribution represented by a QoS ontology model of the transport layer integrating the semantic related to services, protocols, functions and mechanisms. A model-driven architecture based on ontologies is proposed for designing the next generation Autonomic Transport Protocol (ATP) and Autonomic Transport Layer (ATL).
- Chapter 2 presents our second contribution: the component-based Fully Programmable Transport Protocol. This chapter also includes a state of the art on service-oriented and service-component architectures and describes how these approaches can guide the design of the next generation transport layer architecture.
- Section 3 presents our third contribution: the extension of the FPTP protocol named Enhanced Transport Protocol. ETP is an adaptive protocol aimed at implementing behavioral and structural adaptation strategies based on the network environment conditions and guided by the application requirements. Based on the lessons learned from ETP and the benefits offered by the autonomic computing paradigm, several guidelines aimed at applying this paradigm in the design of the next generation transport layer and transport protocols are also presented.

Finally, the conclusions and perspectives of this study are given.

3. Keywords

Keywords: model-driven architecture, ontology-driven architecture, component-based design, service-oriented architecture, autonomic computing, transport protocols, distributed multimedia applications, quality of service.

Chapter I: Transport Layer Modeling

1. Introduction

With the accelerated development of Internet and the diversity of mobile devices (smart-phones, tablets, netbooks and laptops), a new large family of applications and services are available today. Multi-platform instances of a same application are available at home, at work, in our mobile devices or more recently within the Cloud. These applications present heterogeneous needs in terms of Quality of Service (QoS) mainly related with time, bandwidth and reliability requirements. Moreover, networked applications are constantly changing from high-speed and high-bandwidth networks (e.g. ADSL networks at home), to variable bandwidth and high delay networks (e.g. when operating over WiFi or 3G mobile wireless networks).

For traditional applications offering file transfer, web navigation or email functionalities, a fully ordered and fully reliable transport service such as the one offered by TCP over a Best-Effort network is well suited. However, time-constraint applications such as multimedia and interactive applications could prefer a partially reliable and partially ordered service able to offer a more suited lower end-to-end delay. Likewise, current heterogeneous network environments lead to more complex scenarios from the transport layer point of view. Even if traditional protocols such as TCP perform well over classical wired IP networks, their performances are suboptimal under new network technologies. Actually, current high speed, wireless and mobile networks deeply modify the QoS characterization of the network layer by providing more complex service models in terms of bandwidth, losses, delay or jitter.

These new models of the current heterogeneous network layer, as well as the complex requirements demanded by the new generation application layer, have deeply impacted the traditional transport layer. Classical TCP [Postel 1981] and UDP [Postel 1980] protocols are not able to provide an optimal transport service in this new context. This explains the trends for transport protocols creation and extensions proposals during the last decades. Examples of the transport layer evolution are the specializations proposed by the IETF aimed at enhancing traditional transport protocols (e.g. TCP congestion control and error control extensions [Duke 2006], TCP extensions for mobile wireless networks [Inamura 2003], TCP fast retransmission and recovery strategies [Stevens 1997], TCP satellite enhancements [Allman 1999], UDP-lite [Larzon 2004], Reliable UDP [Bova 1999], etc). Likewise, completely new transport protocols such as the Stream Control Transmission Protocol or SCTP [Stewart 2007] and the Datagram Congestion Control Protocol or DCCP [Kohler 2006] have also been developed. Recently, a new IETF group has been created in order to propose an important extension to TCP called MPTCP [Ford 2010] in order to take advantage of the new network capabilities of end-terminals (i.e. multi-homing and multi-paths).

We believe that the current diversity of transport services as well as the complexity resulting from the deployment of a particular transport protocol or transport mechanism over the different services provided by heterogeneous networks ask for a novel design of the transport layer. The next generation transport layer must be able to cope with the diversity

and complexity involved in the new family of transport protocols. Moreover, current and future applications will only be able to take advantage of the most adapted and available transport service if they are able to efficiently interact with this advanced transport layer.

In this chapter a methodology based on a model-driven architecture approach aimed at guiding the design of the next generation transport layer is proposed. This methodology is initiated by the analysis of abstract standard models and protocols and it is guided by the current trends on transport layer evolution. The resulting model has been built following an ontology-driven architecture (ODA) approach. In this approach, a QoS ontology model integrating standard transport services, protocols, functions and mechanisms has been defined in order to represent and characterize the entities, concepts and relationships composing this complex domain. This extensible semantic model is also intended to facilitate the integration of future mechanisms and protocols extensions resulting from the transport layer evolution. Likewise, perspectives of using this model in order to facilitate transport service discovery, selection and composition based on application requirements and network constraints will be presented. Furthermore, the use of this model to guide transport self-managing strategies in dynamic and heterogeneous environments will also be envisaged.

This chapter is structured as following. Section 2 presents a state of the art of the transport layer, including standard service models and traditional transport protocols. Section 3 introduces the model-driven architecture approach and its specialization based on ontology models. Section 4 presents the ontology-driven architecture design represented by a QoS transport ontology model for the new generation transport layer. Finally, the conclusions and perspectives of this chapter are presented.

2. Transport Layer State of the Art

2.1. OSI Model

The Open System Interconnection (OSI) reference model was developed by the International Organization for Standardization (ISO) to provide a conceptual framework for designing and developing interconnected open systems [X200 1994].

The OSI model describes how information from an application in one computer moves through a communication channel to another application in another computer. This model integrates a set of conventions for the definition of services [X210 1993]. An OSI service is defined by the following elements: the service provider, the service user and the service definition. An OSI service definition includes the complete expression of the behavior to be perceived by the service user as well as the service interface. However, service definitions are clearly separated from the mechanisms or protocols implemented by the service provider to offer such service.

The OSI model follows a layered architectural approach composed of seven layers: application, presentation, session, transport, network, data-link and physical layers. Every

layer is described by the set of services offered, the functions implemented within the layer and the services required by the lower layer.

2.2. TCP/IP Model

The TCP/IP model was originally proposed by DARPA in 1970 in order to provide a standard framework for developing services and protocols allowing computers communication. The TCP/IP model or Internet protocol is maintained by the Internet Engineering Task Force (IETF) and proposes 4 layers: link, IP, transport and application layers [Braden 1989], [Braden 1989a].

2.3. Transport layer

The transport layer in the OSI model is aimed at providing transparent transfer of data between the communicating systems and relieve transport service users from the details of using the available network services [X214 1995]. Transport protocols implement the transport functions operating over the network services in order to offer the required transport services to the OSI session layer.

The transport layer in the TCP/IP model is aimed at providing end-to-end communication services for applications. In the original TCP/IP model, only UDP and TCP transport protocols were considered.

Next paragraphs introduce the primary transport services offered by the transport layer as well as a state of the art of the current transport protocols implemented by the TCP/IP transport layer.

2.4. Transport services

Transport services can be characterized as [Iren 1999]:

- Connection-oriented and connectionless. A transport user generally performs three phases of operation: connection establishment, data transfer and connection termination. A connection oriented transport service provides primitives for the three operations and a connectionless service supports only the data transfer operation. Moreover, a connection-oriented service maintains state information about the connection (i.e. messages sequence number, buffer sizes, etc).
- Message-oriented and byte-stream oriented. In the former, users send messages or service data units (SDUs) having a specified maximum size and message boundaries are preserved. In the byte-stream service the data sent by the user is transported as a flow of bytes and messages boundaries are not preserved.
- Blocking and non-blocking. A blocking service ensures that the transport layer is not saturated with incoming data. A non-blocking service allows the sender to submit data and continue operating without taking into account the transport layer buffering capabilities or the rate at which the user receiver consumes data.

- Multicast and unicast. A multicast service enables a sender to deliver data to one or more receivers. An unicast service limits the data delivery to exactly one user receiver.
- Reliable and ordered. A reliable transport service is the one providing guarantees of no losses, no duplicates, order and data integrity.
- Flow and congestion controlled. Transport protocols implementing flow control are able to limit the rate at which data is sent over the network in order to avoid exceeding buffering capacity of the receiver. Transport protocols implement congestion control in order to avoid network congestion collapse.
- Quality of service. A QoS oriented transport protocol is the one providing a service able to take into account application requirements such as end-to-end delay, jitter, throughput, priorities, security, etc.

2.5. Transport protocols

In the original TCP/IP model, the TCP and UDP protocols were designed to operate over the best-effort IP network service. In order to cope with new network technologies, several specializations of these protocols as well as new protocols such as DCCP and SCTP have been developed. These enhancements and new developments mainly propose more adapted error and congestion control for wireless networks as well as new strategies for multi-homing and mobility support. More recently, a new transport protocol named MPTCP is being designed by the IETF in order to enhance TCP and to take advantage of the multi-homing multipath capabilities of end systems.

2.5.1. TCP

The Transmission Control Protocol (TCP) is the most widely used transport protocol. TCP offers an unicast, reliable and in sequence end-to-end data transfer service between two interconnected systems [Postel 1981]. TCP is a connection oriented and byte-stream oriented protocol and implements error detection, error reporting and error recovering mechanisms in order to provide a fully reliable and fully ordered service. Moreover, TCP implements a flow control function in order to avoid exceeding the receiver buffers capacities. Furthermore, TCP implements a congestion control function in order to avoid and react to network congestion.

In order to cope with the constant evolution of the network layer, an important number of enhancements and extensions to the original TCP protocol have been proposed. In [Duke 2006] a description of the major recommendations mainly concerning congestion and error control is presented. Enhancements to the original congestion control protocol of TCP have originated new versions such as TCP Reno, TCP Vegas, TCP New Reno or FAST TCP. Other important recommendations propose extensions in the areas of mobile wireless networks [Inamura 2003], satellite networks [Allman 1999] and other general improvements to the original TCP mechanisms [Allman 1999a], [Stevens 1997], [Mathis 1996], [Jacobson 1992], etc.

2.5.2. UDP

The User Datagram Protocol (UDP) offers an unicast or multicast, connection-less and message-oriented transport service [Postel 1980]. Like TCP, UDP implements a mechanism based on a checksum strategy for detecting and discarding corrupted data, where if a simple bit error is detected at the receiving side, the data packet or datagram is discarded. However, UDP does not include any error or congestion control function. Error, flow and congestion control functions such as the ones implemented by TCP may induce transmission delay and variable throughput. These effects might be not compatible with the requirements of applications such as multimedia applications which demand guarantees on throughput and delay and can tolerate some degree of packet loss. For this reason, these applications have been commonly implemented using UDP in combination with other protocols (i.e. RTP/RTCP) in order to obtain a more adapted transport service [Schulzrinne 2003]. These applications do not always implement congestion control functions and their behavior may be unfriendly and harmful to the rest of applications sharing the same network resources [Floyd 1999].
Several enhancements recommendations have been proposed based on the original UDP, such as UDP lite where partial checksums protect only a part of the datagram [Larzon 2004], or the Reliable UDP or RUDP [Bova 1999].

2.5.3. SCTP

The Stream Control Transmission Protocol (SCTP) offers an unicast, session-oriented and fully reliable transport protocol [Stewart 2007]. In contrast to TCP which is a connection-oriented protocol, SCTP is based on the establishment of a session between the two end-systems. When end-systems present several network addresses, the address list is exchanged during the association establishment phase. Supporting several IP addresses into the same SCTP session is known as multi-homing.

The SCTP error control function detects lost, disordered, duplicated or corrupted packets and uses retransmission schemes to implement the error recovery mechanism. SCTP uses selective acknowledgment or SACK mechanism in order to commit data reception. Retransmission mechanism is performed after a timeout or when the SACK indicates that some SCTP packets have not been received.

SCTP is a message oriented transport protocol. SCTP packets are composed of a common header and data chunks. Multiple chunks may be multiplexed into one packet smaller or equal than the maximum transmission unit (MTU) allowed for the path. A chunk may contain either control information or user data.

SCTP offers a multi-streams service, which means that applicative data can be partitioned in different streams that can be delivered using several independent ordered sequences. Indeed, SCTP does not enforce any ordering constraints between the different streams. It provides a full ordered intra-stream service and an unordered inter-stream service. This service guarantees that if some loss or disordering is detected in one stream then data delivery over the rest of streams is not affected.

In contrast, flow and congestion control functions are implemented on the session basis and not independently for every stream. These functions are based on the TCP algorithms: the receiver informs the sender about the reception buffer capacity and a congestion window is maintained for the SCTP session. The slow-start, congestion avoidance, fast-recovery and fast-retransmission mechanisms are implemented following the TCP algorithms but using the SCTP packets as the acknowledgment unit instead the bytes-acknowledgment approach followed by TCP.

Several extensions have been proposed to SCTP, such as a partial reliability extension [Stewart 2004]. This extension defines the Partial Reliable Stream Control Transmission Protocol (PR-SCTP). PR-SCTP is aimed at providing a partially reliable and fully ordered service to applications transmitting unreliable content.

Other extensions of SCTP include dynamic address reconfiguration [Stewart 2007a], chunks authentication [Tuexen 2007], etc.

2.5.4. DCCP

The Datagram Congestion Control Protocol or DCCP offers a non-reliable transport service for datagram flows regulated by a congestion control function [Kohler 2006]. DCCP is suited to applications currently using UDP. In order to avoid network congestion, applications that use UDP services should implement their own congestion control function. DCCP is intended to deliver a transport service that combines both the efficiency of UDP and the congestion control and network friendliness of TCP.

DCCP allows the negotiation of the congestion control function to be used. DCCP uses a Congestion Control Identifier or CCID to identify the congestion control function to be used for each direction of the DCCP connection. Several congestion control functions have been proposed by the IETF such as the TCP-like congestion control using a congestion window [Floyd 2006], the TCP-friendly rate control or TFRC using an equation to estimate de rate allowed [Floyd 2008] and the TCP-Friendly Rate Control for Small Packets or TFRC-SP [Floyd 2009].

Several enhancements to DCCP have been proposed, such as Quick-Start for DCCP [Fairhurst 2009] or faster restart for TFRC [Kohler 2008]

2.5.5. MPTCP

The Multipath Transport Protocol (MPTCP) offers a set of extensions to TCP aimed at taking advantage of multi-homing capabilities of end-systems in order to increase the efficiency of network resources usage as well as improving resilience to connectivity problems [Ford 2010], [Ford 2010a]. The concurrent use of the network resources available through the various network interfaces allows to increase the resource usage efficiency. Likewise, the resilience is improved by the connectivity offered by the multiple paths usage.

MPTCP offers a basic API compatible with existing TCP based application and an advanced API allowing MTPCP-aware application to express preferences taken into account by MTPCP mechanisms [Scharf 2010].

MPTCP is being designed following a compositional architectural approach. The transport functions to be provided by MPTCP are separated in two semantic layers related with application-oriented and network-oriented transport functions.

New congestion control functions are being designed in order to take into account the specificities of MPTCP [Raiciu 2010]. These new functions should be able to be fair with TCP flows while moving traffic away from congested links.

2.6. Transport functions and mechanisms

Most of the transport protocols are implemented based on common fundamental functions such as connection management, multiplexing/demultiplexing, segmentation/reassembly, etc. [Iren 1999]. The heterogeneity in transport services characterization is mainly achieved by the implementation or not of specific error control, flow control and congestion control functions. Specializations of these functions called QoS-oriented transport functions are intended to take into account explicit QoS requirements of applications in terms of delay, jitter or throughput as well as partial reliability and partial order tolerance.

2.6.1. Error control

Error control functions are implemented based on a combination of mechanisms aimed at protecting user data and control information against loss or damage [Iren 1999]. Error control is performed in two phases: error detection and reporting, and error recovery. Error detection mechanism 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 in order to recover from errors. Error recovery mechanisms can be implemented by retransmission mechanisms (.i.e. Automatic Repeat reQuest or ARQ) or redundancy mechanisms (.i.e. Forward Error Correction or FEC).

2.6.2. Flow and congestion control

Flow control function is performed by the sender entity using a rate control mechanism in order to limit the rate at which data is sent over the network. One of the goals of the flow control is to avoid exceeding the capacity of the transport receiver entity. In [Iren 1999], congestion control function is presented as a specialization of the flow control function with the goal of avoiding and limiting network congestion.
Network congestion is characterized by excessive delay and bursts of losses in delivering data packets. The congestion control function is intended to avoid and limit network congestion and its consequences. There are two types of functions for congestion

prevention: the windows-based and the rate-based functions.
The windows-based congestion control function was originally proposed for the TCP transport protocol [Jacobson 1988]. This function also known as TCP-like congestion control, probes the available network bandwidth by slowly increasing a congestion window limiting the data being inserted in the network by the source. Detection of packet loss is considered as an indication of network congestion and the congestion window is greatly reduced in order to avoid network collapse.
Rate-based fucntions are characterized by the use of an estimation of the available network bandwidth as the allowed transmission rate. Rate-based functions may use information about losses and end-to-end delay in order to calculate the transmission rate. The available network bandwidth can be estimated following a probe-based or a model based approach [Wu 2001].

2.6.3. QoS-oriented control

QoS-oriented transport functions are intended to deliver transport services taking into account the application requirements such as end-to-end delay, jitter or throughput. Standard error control functions are commonly focused in providing a fully ordered and fully reliable data transfer service but without taking into account such application constraints. In congested or lossy network environments, such reliable and ordered service increases the end-to-end delay due to the error recovery mechanisms (e.g. retransmissions), producing a final service not satisfactory with time-constrained applications. However, some of these applications may tolerate a partial reliable or partial ordered service providing a lower end-to-end delay on the same network conditions. The Partial Order Connection protocol (POC) was the first protocol proposing partial ordered and partial reliable services [Amer 1994], [Diaz 1994], [Chassot 1995].

Moreover, standard congestion control functions are mainly focused in preserving network resources by reducing the transmission rate. However, the allowed sending rate may be not compliant with the throughput requirements of applications such as video or audio multimedia applications. Several extensions to standard congestion control functions have been proposed in order to be less aggressive than TCP for time-sensitive applications (e.g. TCP-friend rate control [Floyd 2008]). However, QoS applications constraints are not always taken into account.

In [Exposito 2003], several error control and congestion control functions have been proposed in order to offer a service more compliant with QoS application constraints. These QoS-oriented functions have been implemented as specializations of the standard functions by incorporating mechanisms able to take into account time-constraints, partial reliability tolerance and application data priorities.

2.7. Summary

This section has presented the large diversity of services, protocols, functions and mechanisms available at the current transport layer. The development of these solutions has

not followed standard software engineering practices aimed at facilitating software architectures maintainability and evolution.

Next section introduces the Model Driven Architecture approach which is aimed at facilitating portability, interoperability and reusability when designing complex and heterogeneous software systems.

3. Model and Semantic Driven Architecture

This section introduces the model-driven architecture approach aimed at guiding the design and development of complex and evolving systems. This approach guarantees the portability, the interoperability and the reusability of the final system. A methodology based on this approach and using semantic models will be proposed in order to design the architecture of the transport layer of next generation. In the next paragraphs, the model-driven architecture and the various models used to represent different abstract level views of the designed system will be presented. An extension to this approach named ontology-driven architecture will also be presented.

3.1. Model Driven Architecture

The Model Driven Architecture or MDA approach is based on the separation of the specification of a system from its implementation in any specific platform [MDA 2003], [Mellor 2004], [Kleppe 2003]. Model-driven approach allows the use of models to understand, design, develop and maintain a system architecture. The primary goals of MDA are portability, interoperability and reusability.

The MDA approach follows a process based on abstractions by viewpoints. It means that a system can be modeled by abstracting a set of selected architectural concepts and structuring rules. In this way, simplified viewpoints of the system can be constructed. MDA specifies three viewpoints on a system, a computation independent viewpoint, a platform independent viewpoint and a platform specific viewpoint. Each one of these viewpoints are represented or specified using models.

3.1.1. Computation Independent Model

A computation independent viewpoint focuses on the requirements of the system and its environment, hiding the details related to its structure and internal behavior. The Computation Independent Model or CIM represents this viewpoint. The CIM is also known as the domain model and is built using the semantic associated to the system domain.

3.1.2. Platform Independent Model

A platform independent viewpoint focuses on the operation of the system, hiding the details related to specific platforms. This viewpoint should be the same for different platforms. The Platform Independent Model or PIM represents this viewpoint. Platform independence can be obtained by representing the system as operating in a technology neutral virtual machine.

3.1.3. Platform Specific Model

A platform specific viewpoint results from adapting the platform independent viewpoint to specific details of platform. The Platform Specific Model or PSM represents a system at this viewpoint. A PSM combines the specifications in the PIM with the details of using a particular platform.

3.2. Ontology Driven Architecture

The Ontology Driven Architecture or ODA has been proposed by the W3C in order to promote the use of semantic models or ontologies in the framework of the MDA methodology [ODA 2006]. The use of Semantic Web technologies is intended to naturally extend the MDA framework by defining unambiguous domain vocabularies and by providing model consistency checking and validation capabilities [Berners-Lee 2001]. Semantic web technologies are mainly based on implementations following the RDF [RDF 2004] and OWL [OWL 2004] languages specifications.

Ontologies can be used to represent services allowing declarative functionalities to be deployed, discovered and reused. The use of ontologies can facilitate software development by enabling discovering and composition of existing functions to provide a new functionality rather than construct a completely new solution.

ODA may also facilitate dynamic service composition by enabling the definition of semantic models integrating the agreements that software components expose via their interfaces. These semantic models should include preconditions, post-conditions, and invariant rules aimed at specifying the behavior of components and composition of components.

ODA can be used to build semantic models aimed at supporting specification and design phases. These models can also be integrated within the system implementation by including components identification and descriptions and thus enabling discovering and reuse then during design-time and runtime. These ontology models capture semantic related to properties, relationships, and behavior of components. As an ontology is an explicit conceptual model with formal logic-based semantics, the descriptions of components may be queried or may be checked to avoid inconsistent compositions.

In this section, the model-driven and ontology-driven architectures have been introduced. Next section presents our results in applying this methodology to build a QoS-oriented semantic model aimed at guiding the design and development of an evolutive transport layer of next generation.

4. Design of a QoS transport ontology for the next generation transport layer

As previously introduced, the constant evolution of application and network layers have produced an important impact at the transport layer. As a consequence a large diversity of extensions and enhancements to the traditional protocols as well as the design and

implementation of new transport protocols have deeply complexified the transport layer, making the selection of the adequate transport services a difficult task to be programmed at the application layer. For instance, traditional hard-coded strategies for transport socket selection (e.g. static selection of UDP or TCP service) are not well suited anymore in this dynamic context.

A novel approach is required to easily integrate the dynamicity required for a transport layer of next generation. This new approach should facilitate the selection of services and could allow the dynamic deployment of the required transport mechanisms and functions. Due to the complexity related with the diversity of services, protocols, functions and mechanisms, an important effort of semantic characterization and representation is required. In order to apply the Model Driven Architecture approach, next paragraphs introduce a standard framework aimed at providing a referential semantic model for the quality of service. Based on this referential model, we have defined a QoS ontology transport model integrating the semantic of transport requirements, services, protocols, functions and mechanisms for the next generation Transport Layer.

4.1. Quality of service framework

4.1.1. Quality of service definition

Quality has been defined by the ISO-9000 standard as “the degree to which a set of inherent characteristics fulfills requirements”.

Quality of Service (QoS) is defined by the ITU-T Recommendation X.902 as the “set of qualities related to the collective behavior of one or more objects” [X902 1995]

In spite of the diversity of approaches, most of them agree in defining the QoS from the user and service provider viewpoints. The ITU-T Recommendation E.800 [E800 1994] explicitly introduces the user/service approach and defines the QoS as “the collective effect of service performance which determines the degree of satisfaction of a user of the service”. On the one hand, user requirements express the quantitative and qualitative characteristics expected of a particular service; these requirements can generally be expressed in terms of QoS parameters (e.g. time constraints, synchronization, throughput, reliability, order, etc.). On the other hand, from the service provider viewpoint, the QoS is considered as a statement of the level of quality expected to be offered to the user of the service. In ITU-T G. 1000 [G1000 2001] this concept is enhanced integrating two temporal phases: the initial phase where requirements and services can be expressed and the operation phase where the resulting QoS can be observed.

4.1.2. ITU-T X.641 framework

The ITU-T recommendation X.641 has proposed a QoS framework intended to develop standards related to QoS in the area of information technology [X641 1997]. The X.641 framework provides definitions and inter-relationships between these definitions in order to supply a common context for defining, representing and expressing QoS. This framework

introduces the concepts of service, service user, service provider, QoS characteristic, QoS requirement, QoS parameter, QoS management function and QoS mechanism.

Figure 1 illustrates the main concepts introduced by X.641.

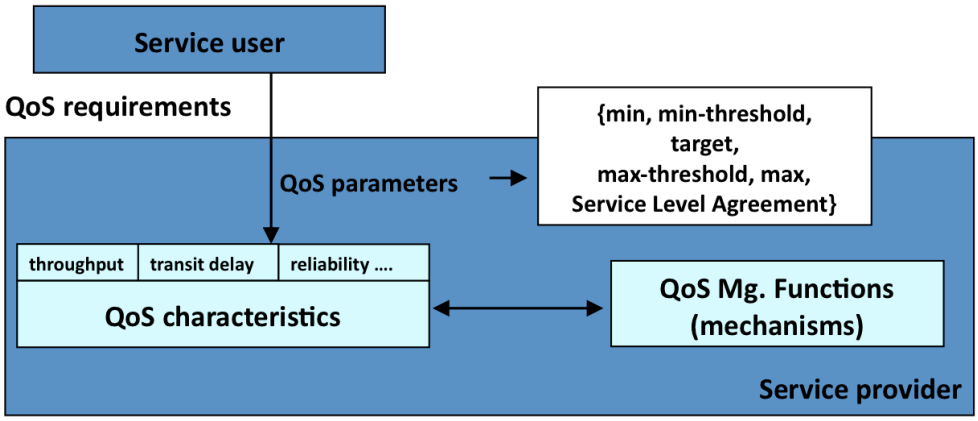

Figure 1. ITU X.641 QoS framework

4.1.2.1. Service

In the ITU-T X.641 framework, service is a very general term that can be applied to the provision of functions such as processing, storing, transmitting, delivering, etc. A service is provided by a service provider to a service user.

4.1.2.2. Service user

A service is delivered to the service user. This user may have QoS requirements such as the maximum delay tolerated to transmit data. These QoS requirements are expressed as QoS parameters that are conveyed to the service provider.

4.1.2.3. Service provider

A service provider is the entity responsible to deliver a service to the service user. The QoS parameters describing the QoS requirements of the service user are conveyed to the service provider. The service provider analyzes the service user requirements and determines the QoS management functions and mechanisms that are required to meet them. The QoS parameters can be conveyed to other entities involved in providing the service. These parameters could be used to produce more detailed QoS requirements to be conveyed to other entities as QoS parameters.

4.1.2.4. QoS characteristic

A QoS characteristic is defined as a quantifiable aspect of QoS of a system, service or resource that is defined independently of the means by which it is represented or controlled. QoS characteristics are intended to be used to model the actual, rather than the observed, behaviour of the systems that they characterize. QoS characteristics definitions include

name, description, quantification unit and optionally statistical derivations and specializations. Examples of QoS characteristics related to communication services are throughput, delay, jitter, order or reliability.

4.1.2.5. QoS requirement

A QoS requirements expresses part or all of the user requirement expected on one or several QoS characteristics.

4.1.2.6. QoS parameter

A QoS parameter is a vector of scalar values describing a QoS requirement in terms of:

- a desired or target level of characteristic
- a maximum or minimum level of a characteristic
- threshold values enabling warning or alert signals to be triggered or operations to be executed
- a measured value, used to convey historical information
- and finally the nature of the service level agreement concerning the parameter.

The term service level agreement is used to describe the nature of the commitment of the service provider to deliver the service required by the service user. The agreement nature determines the actions that the service provider and/or the service-users agrees to take to maintain agreed levels of QoS. Examples of service level agreements are best effort, compulsory or guaranteed.

- Best effort is the weakest agreement indicating that there is no assurance that the agreed QoS will be provided. The service provider is not supposed to perform any remedial action to deliver the QoS to the service user
- Compulsory agreement indicates that the service must be aborted if the QoS degrades below the agreed level. The service provider does not guarantee the QoS required by the service user
- Guaranteed agreement indicates that the service provider must guarantee the QoS required by the service user, and that the service will not be initiated unless it can be maintained within the specified QoS parameters.

4.1.2.7. QoS function

QoS management refers to the activities related to the control and administration of QoS within a system or network. QoS management functions are designed to assist in satisfying one or more user QoS requirements. QoS functions are composed by one or several QoS mechanisms.

4.1.2.8. QoS mechanism

QoS mechanisms are intended to support establishment, monitoring, maintenance, control, or enquiry of QoS. QoS mechanisms are driven by users' QoS requirements expressed as

QoS parameters. These mechanisms commonly operate in collaboration with other QoS mechanisms.

5. Design of a QoS transport ontology for the next generation transport layer

Based on the OSI, TCP/IP and ITU models, a QoS transport ontology integrating requirements, parameters, services, protocols, functions and mechanisms has been defined. This ontology is aimed at providing an unambiguous transport layer vocabulary and enabling managing different levels of representation and validation. This model will be presented in the next paragraphs.

5.1. X.641 QoS ontology

The first step in designing the QoS transport ontology is the definition of the QoS basis of this semantic model. These basis are provided by the generic QoS framework provided by the ITU X.641 recommendation. Figure 2 presents a QoS ontology model that integrates the concepts and relationships proposed by this framework.

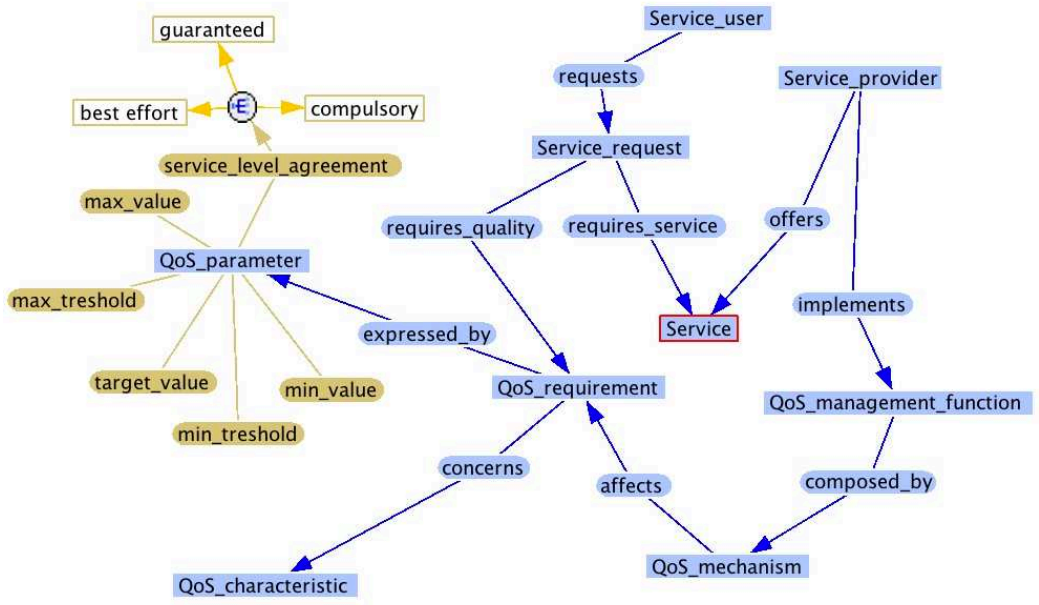

Figure 2. QoS ontology

Note: Figure 3 presents a legend aimed at helping reading the ontologies presented in this document. The visual representation of classes, individuals (or objects), object properties (or relationships between individuals), data properties (or object attributes) and data values is presented. Other elements representing enumerations (E), unions (∪), intersections (∩), compliment (¬), universal (∀) and existential (∃) expressions could also be found int the visual representation of ontologies included in this document.

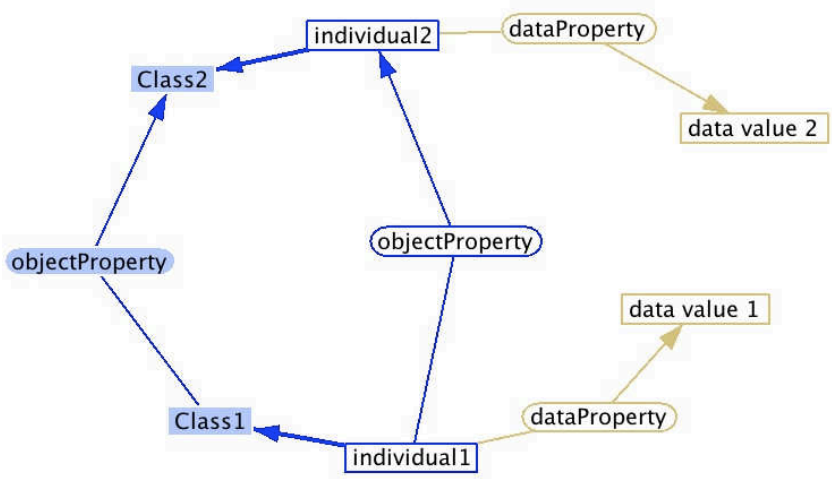

Figure 3. Ontology visual representation legend

This basic QoS ontology will be extended in order to incorporate requirements, mechanisms, functions, protocols and service concepts from the transport layer point of view.

5.2. QoS transport requirements

The expression of application requirements in terms of QoS parameters at the transport layer is built based on the following QoS characteristics:

- Reliability: packet loss rate (PLR) tolerance.
- Order: out of sequence tolerance
- Throughput: transmission capacity per time unit
- Delay: end to end transmission time
- Jitter: variation of the delay

Based on these characteristics, QoS requirements can be expressed in terms of QoS parameters. For instance, for interactive video conferencing applications, examples of parameters expressions for QoS transport requirements are [G1010 2001]:

- Minimum and target values: reliability requirements could be expressed by a minimum value of 97% (e.g. 3% of packet loss rate tolerance) and a target value of 100%.
- Maximum and target values: delay requirements could be expressed by a maximum delay of 400 ms and a target value of 150 ms.
- Service level agreements: best effort agreements could be expressed for all the requirements. Likewise, different agreements could be expressed for each requirement, for example best-effort agreement for reliability and compulsory agreement for delay (e.g. the service should be stopped when the delay exceed the maximum value).

5.3. QoS transport mechanisms, functions and protocols

Based on the state of the art on transport protocols, the various mechanisms implemented by TCP, UDP, SCTP, DCCP and MPTCP protocols have been integrated in the QoS transport ontology. Likewise, transport functions including basic functions (e.g. connection management, multiplexing/demultiplexing, etc.), advanced functions (e.g. multi-streaming and multi-path management), error control functions (e.g. ARQ or FEC) and flow and congestion control functions (e.g. window-based congestion control, TFRC, etc.) have also been integrated in this ontology. Likewise, the various mechanisms implementing these functions have also been incorporated.

5.4. QoS transport ontology

Figure 4 represents the QoS transport ontology integrating application requirements as well as the transport mechanisms, functions and protocols.

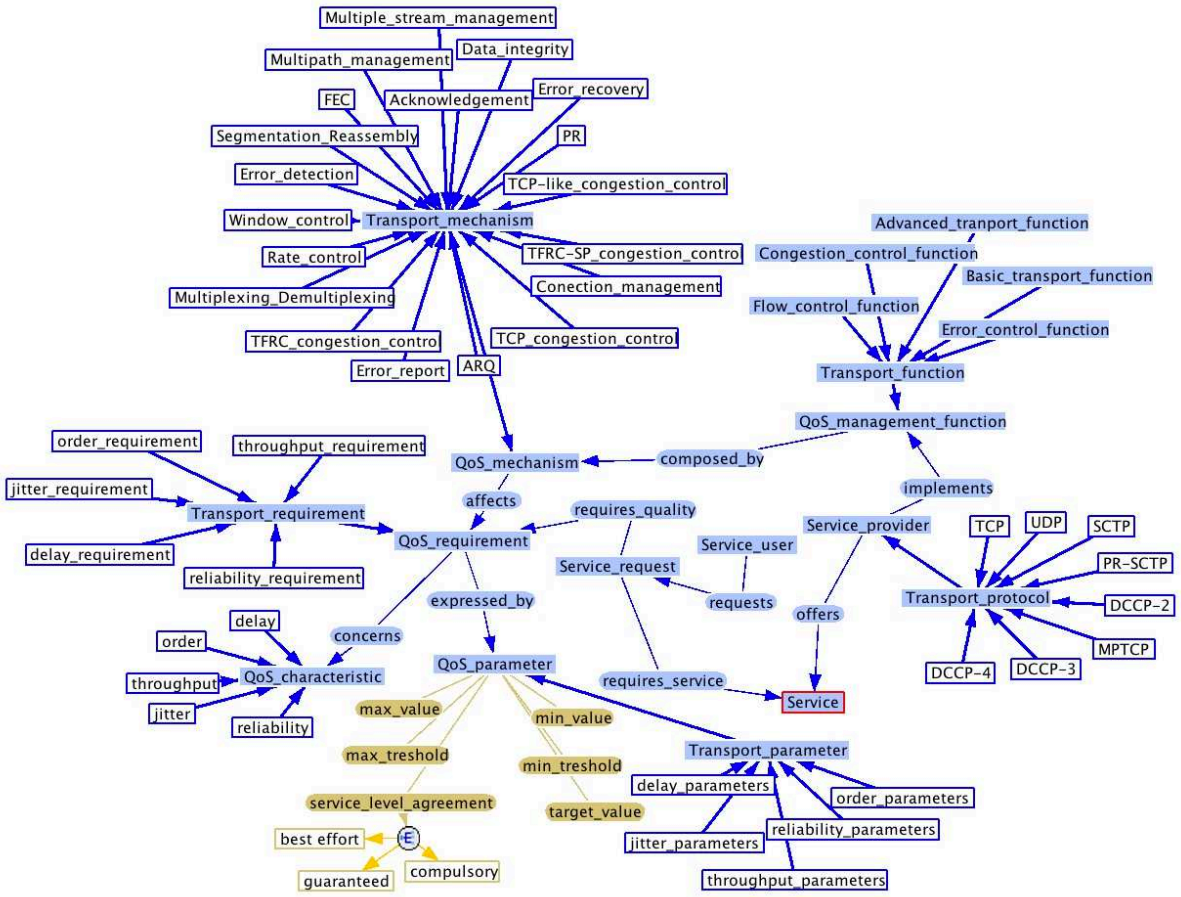

Figure 4. QoS transport ontology

5.5. QoS transport services characterization

Based on the main functions provided at the transport layer, the following characterization of transport services has been defined:

- Error controlled: integrating fully reliable, partially reliable, fully ordered and partially ordered.
- Throughput controlled: includes congestion, flow and rate controlled services.
- Time controlled services: integrates delay and jitter controlled services. This class integrates the services implemented by QoS-oriented transport functions based on specializations of error-control and throughput control functions.

This classification allows to characterize the service offered by the transport protocols as following

- TCP: fully reliable, fully ordered, congestion-controlled and flow-controlled, time-uncontrolled
- UDP: error-uncontrolled (unreliable, unordered), throughput-uncontrolled, time-uncontrolled
- SCTP: fully reliable, fully ordered and unordered, congestion-controlled, flow-controlled, time-uncontrolled. As SCTP offers a multi-stream transport service, it can be considered as intra-stream fully ordered as TCP but inter-stream unordered service.
- PR-SCTP: partially reliable, fully ordered and unordered, congestion-controlled, flow-controlled, time-uncontrolled
- MPTCP: fully reliable, fully ordered, congestion-controlled, flow-controlled, time-uncontrolled
- DCCP-2, DCCP-3 and DCCP-4: error-uncontrolled (unreliable, unordered), congestion-controlled, time-uncontrolled.

Figures 5, 6 and 7 illustrate this transport service classification for error-based, throughput-based and time-based service classifications respectively. These classifications present partial views of the QoS transport ontology shown in the Figure 4.

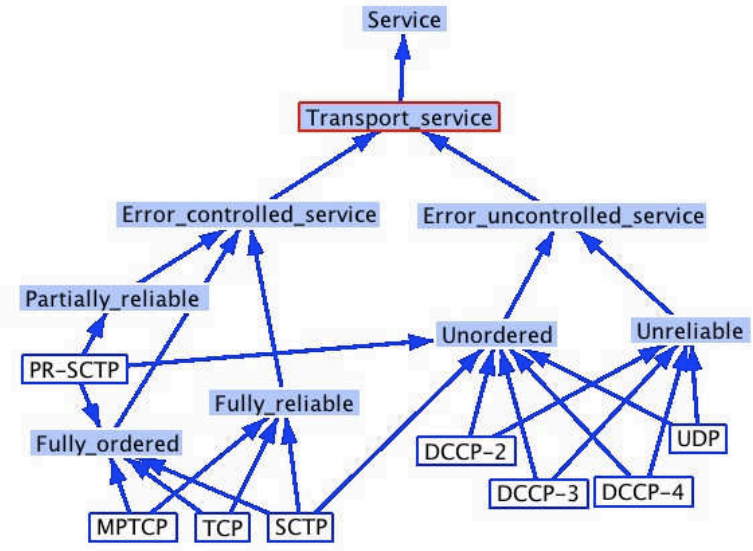

Figure 5. Error-based transport services classification

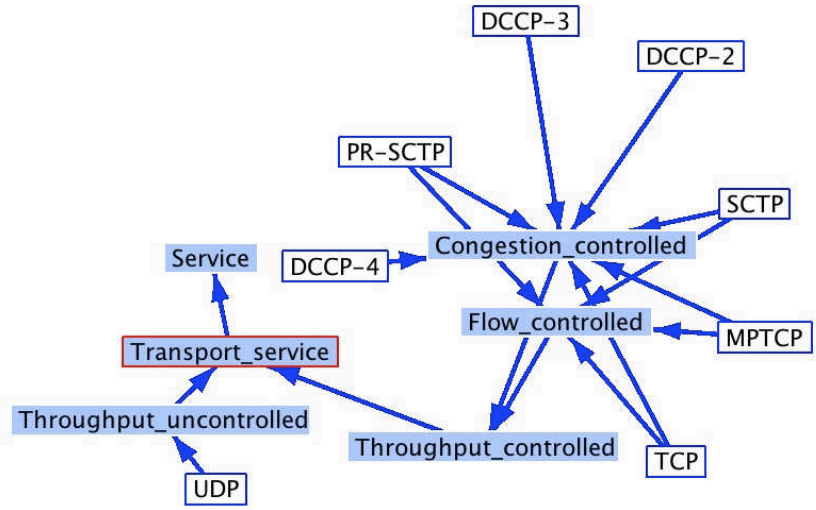

Figure 6. Throughput-based transport services classification

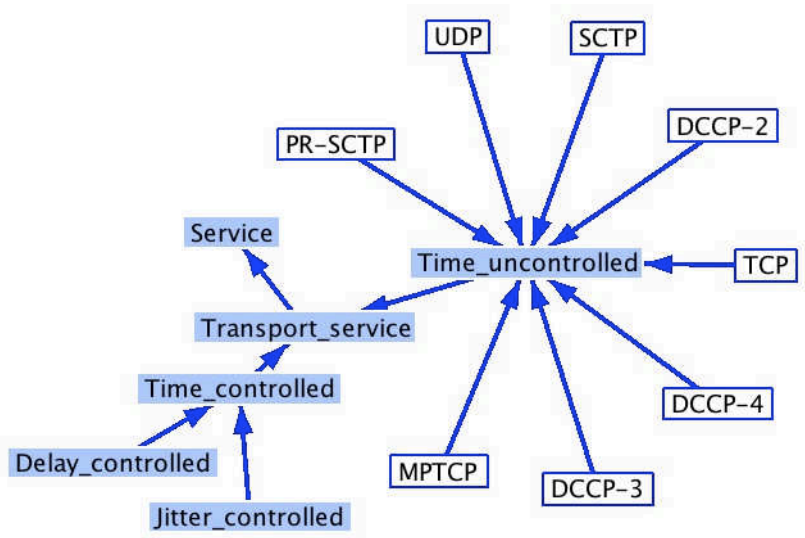

Figure 7. Time-based transport services classification

This service classification could largely facilitate the dynamic selection of the adequate transport service based on application requirements. Indeed, a service-oriented approach based on the expected service properties could be implemented by using this transport ontology. Moreover, the selection of these services could be done dynamically based on the specific platform transport protocols available in the user system. Furthermore, this selection could integrate both the functional and non-function properties of the expected transport service. In the next chapter, the use of this ontology to design and implement a service-oriented transport architecture will be presented.

5.6. Transport components and transport composite characterization

Most of the previously presented transport protocols are based on implementations where mechanisms offering different functionalities (i.e. error control or congestion control) are merged within a same monolithic implementation. However a component-based approach can be followed to characterize them. Actually, each transport protocols can be represented as the implementation of one or several transport functions. The composition relationship between functions has been integrated in this ontology. Figure 8 illustrates this composite approach in the representation of the TCP transport protocol functions.

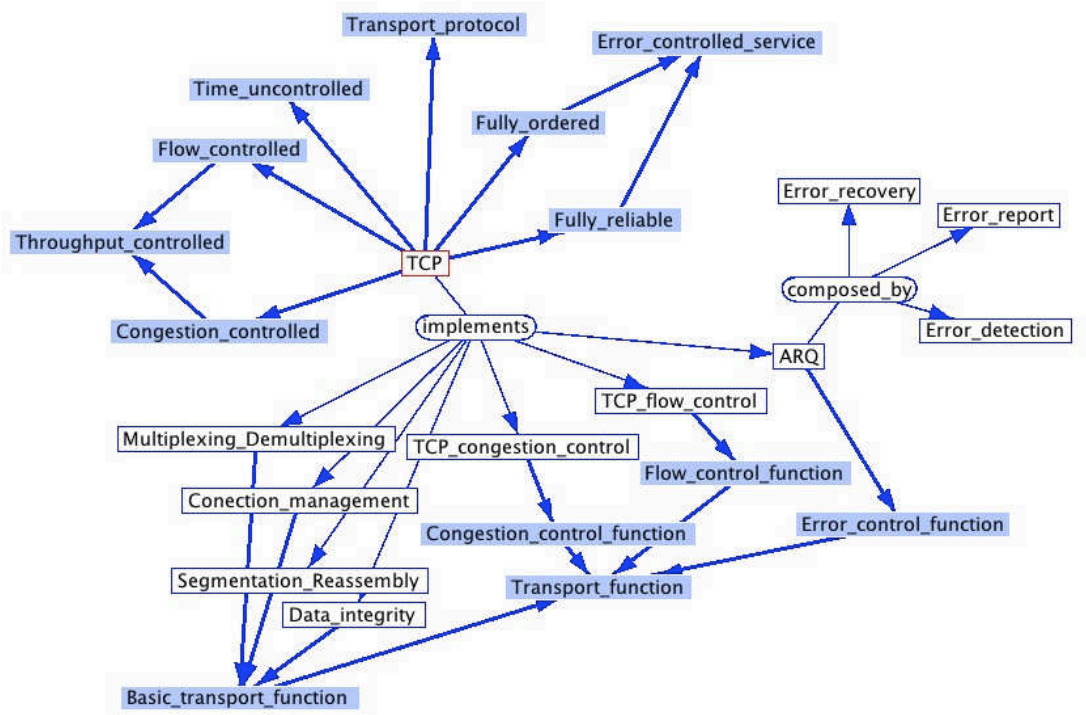

Figure 8. Example of composite-based approach for the TCP transport functions

Likewise, transport functions can be represented as the implementation of one or several transport mechanisms. The composition relationship between functions and mechanisms has also been integrated in the ontology. Figure 9 illustrates how the ARQ error control function and the TFRC congestion control function are implemented as a composition of mechanisms. Both functions share common transport mechanisms (i.e. error detection and error report). However, the specificities of each function are achieved by the addition of an error recovery mechanism for ARQ and a rate control mechanism for TFRC.

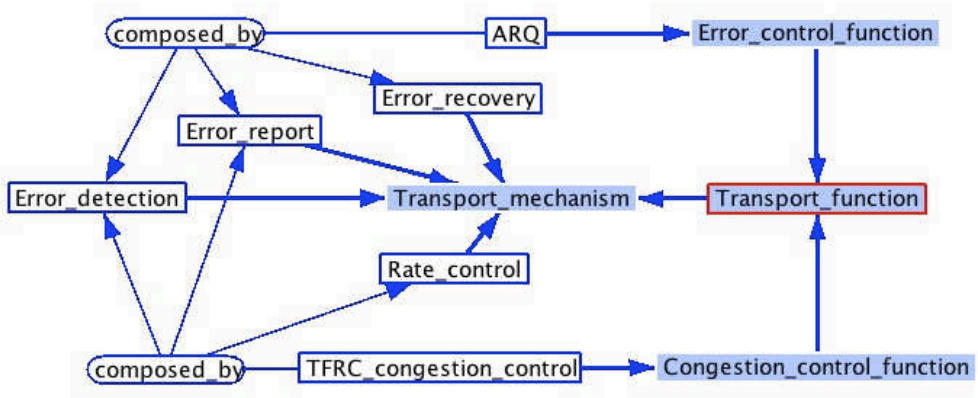

Figure 9. Example of a component-based approach for ARQ error control and TFRC congestion control functions

The use of such component-based approach could widely facilitate the design and development of new transport protocols. Indeed, new transport services could result of the dynamic combination of pluggable components offering the services properties required by the applications.

6. Conclusions and Perspectives

This paper has presented the state of the art of the large diversity of services, protocols, functions and mechanisms available at the transport layer. A methodological approach based on model-driven architecture and adapted to the use of ontology models has also been presented.

In order to better characterize the diversity and complexity involved within the transport layer, a Computation Independent Model (CIM) providing an abstract and high-level service model has been presented based on standard and referential models. The OSI and TCP/IP models have provided the service basis for this CIM model. Likewise, the quality of service basis for the CIM model has been provided by the ITU X.641 framework. Based on this abstract model, a Platform Independent Model (PIM) of the transport layer has been elaborated and its semantic representation based on ontologies has been presented. This PIM model is intended to characterize and classify the large diversity of available transport services, protocols, functions and mechanisms. Likewise, this model provides an unambiguous transport layer vocabulary and enables model consistency checking and validation capabilities.

This ontology model offers the required semantic basis for managing different levels of representation and for integrating current and future services to be offered by the next generation transport layer.

The proposed methodology also provides the basis for developing a service-oriented and component-based architecture for transport protocols. Systems designed and developed following MDA and ODA approaches will gain major benefits in terms of flexibility and extensibility by integrating a service oriented approach. Indeed, Service Oriented Architecture (SOA) approach can be used for designing applications focused on services composition and coordination which is the specification level required by PIM models. Likewise, component-based approach facilitates the discovery and dynamic composition of reusable components, which can satisfy the service specification required for platform specific models. Furthermore, the QoS ontology model can be enhanced in order to integrate the required semantic to design and develop the required self-managing functionalities of autonomic transport protocols.

In the next chapter our work in designing and developing a QoS-oriented and component-based transport protocol will be presented. Likewise, the use of the QoS transport ontology in order to design a service-oriented and component-based architecture for the next generation transport layer will also be presented.

7. Annexes

The ontology defined in this study can be found at

- http://homepages.laas.fr/eexposit/hdr/QoSOntology_V1.owl
- Version txt at http://homepages.laas.fr/eexposit/hdr/QoSOntology_V1.txt

8. References

[Allman 1999] M. Allman, D. Glover, L. Sanchez, Enhancing TCP Over Satellite Channels using Standard Mechanisms, RFC 2488, January, 1999

[Allman 1999a] M. Allman, V. Paxson, W. Stevens, TCP Congestion Control, RFC 2581, April, 1999

[Amer 1994] Paul Amer, Christophe Chassot, C. Connolly, P. Conrad, Michel Diaz, Partial Order Transport Service for Multimedia and other Applications, IEEE ACM Transactions on Networking, 2, 5, October, 1994

[Berners-Lee 2001] O. Lassila, J. Hendler, T. Berners-Lee, The Semantic Web, Scientific American, May, 2001

[Bova 1999] T. Bova, T. Krivoruchka, RELIABLE UDP PROTOCOL, INTERNET-DRAFT, February, 1999

[Braden 1989] R. Braden, Requirements for Internet Hosts -- Communication Layers, RFC 1122, October, 1989

[Braden 1989a] R. Braden, Requirements for Internet Hosts -- Application and Support, RFC 1123, October, 1989

[Chassot 1995] Christophe Chassot, Architecture de Transport Multimédia à Connexions d'Ordre Partiel, Institut National Polytechnique de Toulouse, December, 1995

[Diaz 1994] Paul Amer, Christophe Chassot, André Lozes, Michel Diaz, Partial Order Connections: A new concept for High Speed and Multimedia Services and Protocols, Annals of Telecommunications, 49, 5-6, May-June, 1994

[Duke 2006] M. Duke, R. Braden, W. Eddy, E. Blanton, A Roadmap for Transmission Control Protocol (TCP) Specification Documents, RFC 4614, September, 2006

[E800 1994] ITU-T Rec. E.800, Terms and definitions related to Quality of Service and Network Performance including dependability, August, 1994

[Exposito 2003] Ernesto Exposito, Specification and implementation of a QoS oriented transport protocol for multimedia applications, Ph.D. dissertation, Université de Toulouse, Institut National Polytechnique de Toulouse, December 17, 2003

[Fairhurst 2009] G. Fairhurst, A. Sathiaseelan, Quick-Start for the Datagram Congestion Control Protocol (DCCP), RFC 5634, August, 2009

[Floyd 1999] S. Floyd, K. Fall, Promoting the use of end-to-end congestion control in the Internet, IEEE/ACM Transactions on Networking, 7, 4, August, 1999

[Floyd 2006] S. Floyd, E. Kohler, Profile for Datagram Congestion Control Protocol (DCCP), Congestion Control ID 2: TCP-like Congestion Control, RFC 4341, March, 2006

[Floyd 2008] S. Floyd, M. Handley, J. Padhye, J. Widmer, TCP Friendly Rate Control (TFRC): Protocol Specification, RFC 5348, September, 2008

[Floyd 2009] S. Floyd, E. Kohler, Profile for Datagram CongestionControl Protocol (DCCP), Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP), RFC 5622, August, 2009

[Ford 2010] A. Ford, C. Raiciu, S. Barre, J. Iyengar, Architectural Guidelines for Multipath TCP Development, IETF Internet-draft, June, 2010

[Ford 2010a] A. Ford, C. Raiciu, M. Handley, TCP Extensions for Multipath Operation with Multiple Addresses, IETF draft, July, 2010

[G1000 2001] ITU-T Rec. G.1000, Communications quality of service: A framework and definitions, November, 2001

[G1010 2001] ITU-T Rec. G.1000: End-user multimedia QoS categories, November, 2001

[Inamura 2003] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, F. Khafizov, TCP over Second (2.5G) and Third (3G) Generation Wireless Networks, RFC 3481, February, 2003

[Iren 1999] Sami Iren, Paul D. Amer, P. Conrad, The transport layer: tutorial and survey, ACM Computing Surveys, 31, 4, 1999

[Jacobson 1988] V. Jacobson, M. Karels, Congestion avoidance and control, SIGCOMM '88: Symposium proceedings on Communications architectures and protocols, New York, NY, USA, August, 1988

[Jacobson 1992] V. Jacobson, R. Braden, D. Borman, TCP Extensions for High Performance, RFC 1323, May, 1992

[Kleppe 2003] Anneke Kleppe; Jos Warmer; Wim Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley Professional, April, 2003

[Kohler 2006] E. Kohler, M. Handley, S. Floyd, Datagram congestion control protocol (DCCP), RFC 4340, March, 2006

[Kohler 2008] E. Kohler, S. Floyd, A. Sathiaseelan, Faster Restart for TCP Friendly Rate Control (TFRC), RFC draft-ietf-dccp-tfrc-faster-restart-06.txt, July, 2008

[Larzon 2004] L-A. Larzon, M. Degermark, S. Pink, L-E. Jonsson, G. Fairhurst, The Lightweight User Datagram Protocol (UDP-Lite), RFC 3828, July, 2004

[MDA 2003] MDA Guide Version 1.0.1, OMG, June, 2003

[Mathis 1996] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgment Options, RFC 2018, October, 1996

[Mellor 2004] Stephen J. Mellor; Kendall Scott; Axel Uhl; Dirk Weise, MDA Distilled: Principles of Model-Driven Architecture, Addison-Wesley Professional, March, 2004

[ODA 2006] Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering, W3C Working Draft, April, 2006

[OWL 2004] OWL Web Ontology Language Overview, W3C Recommendation, February, 2004

[Postel 1980] Postel Jon, User Datagram Protocol (UDP), RFC 768, August, 1980

[Postel 1981] Postel Jon, Transmission Control Protocol, DARPA Internet Program Protocol Specification, RFC 793, September, 1981

[RDF 2004] Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, February, 2004

[Raiciu 2010] C. Raiciu, M. Handley, D. Wischik, Coupled Multipath-Aware Congestion Control, IETF draft, July, 2010

[Scharf 2010] M. Scharf, A. Ford, MPTCP Application Interface Considerations, IETF draft, July, 2010

[Schulzrinne 2003] Schulzrinne H., Casner S., Frederick R., And Jacobson V., RTP: A Transport Protocol for Real-Time Applications, RFC 3550, July, 2003

[Stevens 1997] W. Stevens, TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC 2001, January, 1997

[Stewart 2004] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, P. Conrad, SCTP Partial Reliability Extension, RFC 3758, May, 2004

[Stewart 2007] R. Stewart, Ed., Stream Control Transmission Protocol, RFC 4960, September, 2007

[Stewart 2007a] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, M. Kozuka, Stream Control Transmission Protocol (SCTP), Dynamic Address Reconfiguration, RFC 5061, September, 2007

[Tuexen 2007] M. Tuexen, R. Stewart, P. Lei, E. Rescorla, Authenticated Chunks for the Stream Control Transmission Protocol (SCTP), RFC 4895, August, 2007

[Wu 2001] D. Wu, Y. Hou, W. Zhu, Zhang, J. Peha, Streaming Video over the Internet: Approaches and Directions, IEEE Transactions on Circuits and Systems for Video Technology, 11, 3, March, 2001

[X200 1994] ITU-T Recommendation X.200, Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model, July, 1994

[X210 1993] ITU-T Recommendation X.210, Information technology - Open Systems Interconnection - Basic Reference Model: Conventions for the definition of OSI services, November, 1993

[X214 1995] ITU-T Recommendation X.214, Information technology - Open Systems Interconnection - Transport Service Definition, november, 1995

[X641 1997] ITU-T Information Technology - Quality of Service Framework, ITU-T X.641 (ISO/IEC IS13236), December 17th, 1997

[X902 1995] ITU-T Rec. X.902, Information technology - Open Distributed Processing – Reference Model: Foundations, November, 2005

Chapter II: Component-based and Service-oriented Transport Layer

1. Introduction

In the previous chapter, the large diversity of transport services, protocols, functions and mechanisms have been presented. The complexity involved in the current transport layer will become even more important in the next years as experts in the transport layer such as the IETF working groups keep working on designing and implementing new solutions motivated by new applications and the dynamic evolution of the network layer. The transport layer of the next generation requires a modern software architecture able to facilitate applications service usage while assuring smooth and efficient integration of future mechanisms and protocols.

The requirements guiding the design of the next generation transport layer architecture can be expressed from two points of view:

- From the application programmer point of view: the transport layer needs to provide dynamic services discovery, deployment and binding capabilities. In this way, during development-time, an application programmer only needs to request the required service or to express the expected services properties, thus concentrating his/her efforts into the application business logic. During runtime, applications will bind to the most adequate transport service available on the user terminal based on the specific network environment.
- From the transport protocol programmer point of view: the transport layer has to provide an extensible and pluggable architecture able to easily integrate new transport components while assuring the interoperability with existing transport protocols. In this way, protocol programmers can develop new components such as for instance forward error control mechanisms for wireless networks or time-constrained retransmission mechanisms for delay-tolerant applications. These new components could easily be integrated within a pluggable software architecture allowing new transport services to be automatically discovered, composed and used by the applications with a minimum effort. Moreover, these components could be designed as platform-independent components and during the deployment phase the most adequate platform-specific components could be downloaded and deployed within the transport layer of the networked device. This kind of dynamic deployment approach guarantees that the most recent and adapted component release is always used.

In this chapter, our work in designing and developing a component-based transport protocol will be presented. This protocol provides an extensible architecture for composing mechanisms intended to satisfy a large diversity of application requirements over heterogeneous network service. Based on the lessons learned in developing this protocol and the methodology introduced in the previous chapter, solutions in the area of service-oriented, component-based and semantic-driven architectures will be explored in order to provide guidelines and perspectives for designing the transport layer of the next generation.

This chapter is structured as following: section 1 presents a state of the art and related works on software architectures aimed at designing and developing communication and component-based protocols. Section 2 presents our first contribution in this area represented by the Fully Programmable Transport Protocol (FPTP), its architecture, its components and several evaluations studies. Section 3 presents a state of the art of the service-oriented and component-based architectural paradigms. Section 4 presents our second contribution, represented by use of the QoS transport ontology model to guide the selection and composition of services for the next generation transport layer. Finally, several conclusions are presented.

2. Architectural frameworks for communication protocols

During last years, different compositional frameworks have been proposed to design and implement communication architectures. Most of these frameworks follow a hierarchical compositional model. Hierarchical composite architectures provide a model where a communication system is designed as a stack of directed graph of components exchanging data or control messages. Examples of these hierarchical architectures are represented by the V_STREAMS [Ritchie 1984] and the X-kernel [Hutchinson 1991] models.

Other architectural frameworks are based on an event-base model. Event-based composite architectures promote a non-hierarchical compositional model, where there is no mandatory sequential order between components composing the system. The ADAPTIVE [Schmidt 1993] and Cactus [Wong 2001] systems are examples of event-based architectures.

Next paragraphs present a state of the art of these architectures.

2.1. Hierarchical architectures

2.1.1. V-STREAMS

The V_STREAMS proposes a hierarchical and compositional model for the input-output stream system connecting an user's program to a device within Unix systems [Ritchie 1984]. In this system, a stream is a full duplex connection between several linearly connected processing modules with data flowing in both directions.

The modules in a stream communicate almost exclusively by passing messages to their neighbours, except for some conventional variables used for some control functions. The API services consist in writing and reading 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.

Figure 1 presents the V_STREAM hierarchical architecture.

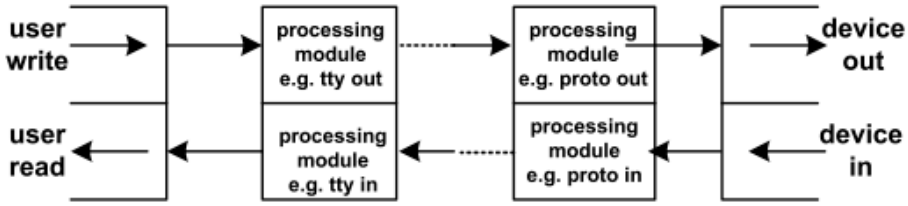

Figure 1. V_STREAM hierarchical architecture

2.1.2. X-kernel

The X-kernel proposes a hierarchical architecture model based on a directed protocol graph implemented in an object based framework [Hutchinson 1991]. In the X-kernel framework, a protocol is considered as an abstract object that exports both a service interface and a peer-to-peer interface. The former defines the operations by which other protocols on the same machine invoke the services of this protocol, while the latter defines the form and meaning of the messages exchanged between peers to implement the service. X-kernel supports an interesting mechanism to configure a protocol stack. This mechanism is based on a protocol graph, which makes it easy to plug protocols together in different ways. This protocol graph is statically configured at initialization time.

Figure 2 illustrates the X-Kernel hierarchical architecture.

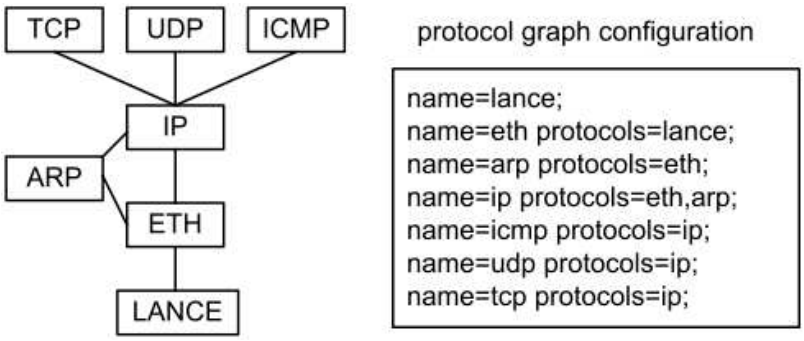

Figure 2. X-Kernel hierarchical architecture

2.2. Event-based architectures

2.2.1. ADAPTIVE

ADAPTIVE proposes A Dynamically Assembled Protocol Transformation, Integration, and eValuation Environment for developing and experimenting with flexible transport system architectures [Schmidt 1993]. This architecture implements a set of fundamental design patterns that simplify the development of concurrent event-driven communication software following an event-based approach.

ADAPTIVE automates communication software configuration and reconfiguration by dynamically linking services into applications at run-time and executing these services on one or more processes or threads. These services are implemented by a collection of object-

oriented components. ADAPTIVE provides a higher level configuration interface, where a protocol composition is created automatically based on a functional specification.

2.2.2. Cactus

Cactus is a framework for constructing configurable network services [Wong 2001]. In Cactus each service attribute is implemented as an independent software module called a micro-protocol. In this approach, micro-protocols are chosen based on the requirements of the higher levels that use the service or on the specific characteristics of the underlying network or computing platform.

The configurable transport protocol (CTP) has been designed using the Cactus framework. In CTP various attributes such as reliable transmission and congestion control are implemented as separate micro protocols, which are then combined in different ways to provide customized semantics.

2.3. Summary

Hierarchical and event-based architectural models have allowed us to design and develop the architecture of a component-based transport protocol able to allocate the different mechanisms and functions required to offer a large diversity of services based on application requirements and network constraints. Next paragraphs present this protocol named Fully Programmable Transport Protocol.

3. The Fully Programmable Transport Protocol (FPTP)

The Fully Programmable Transport Protocol or FPTP is a message-oriented transport protocol offering a partially reliable error-controlled, congestion-controlled and timed-controlled transport service [Exposito 2003a]. FPTP follows the Partial Order Connection approach (POC) in order to offer a more flexible error control function able to implement fully reliable or partially reliable services. FPTP has been designed to be configured based on the application requirements and the underlying network services. FPTP services are implemented by the composition of mechanisms suited to control and manage the QoS. The architecture proposed by FPTP allows a designer to enhance the basic proposed mechanisms in order to provide a larger family of transport services. The design principles, the compositional architecture and the programmable and configurable mechanisms provided by FPTP will be introduced in next paragraphs.

3.1. FPTP composite architecture

FPTP promotes a hierarchical and event-based compositional architectural model intended to the deployment of transport mechanisms at two different planes [Exposito 2005]:

- data or control plane: functions operating at the control plane are implemented by the compostion of mechanisms operating synchronously to the ADUs exchanged between the applications (e.g. source, receiver and hybrid-based mechanisms as for

instance flow control, error detection, etc.). Mechanisms operating at the control plane are deployed following a hierarchical architecture approach.

- management plane: The management plane includes transport functions based on mechanisms performed asynchronously to the data flows, for instance when a specific event is triggered (e.g. retransmission mechanism triggered by a timeout). Management functions can be implemented as a composition of mechanisms following a hierarchical and an event-based architecture (such as the proposed by ADAPTIVE or Cactus). Moreover, communication channels can be established between the control and management planes to allow the exchange of signals between mechanisms operating in both planes. These signals are intended to send management orders and receive monitoring indicators in order to implement the adequate management functions.

In summary, FPTP promotes an architecture following a hierarchical model for the composition of mechanisms related to functions operating at the control plane and an hybrid model (i.e. hierarchical and event-based) for the functions operating at the management plane. Figure 3 presents an UML composite diagram specifying the composite communication pattern of FPTP [Exposito 2004a]

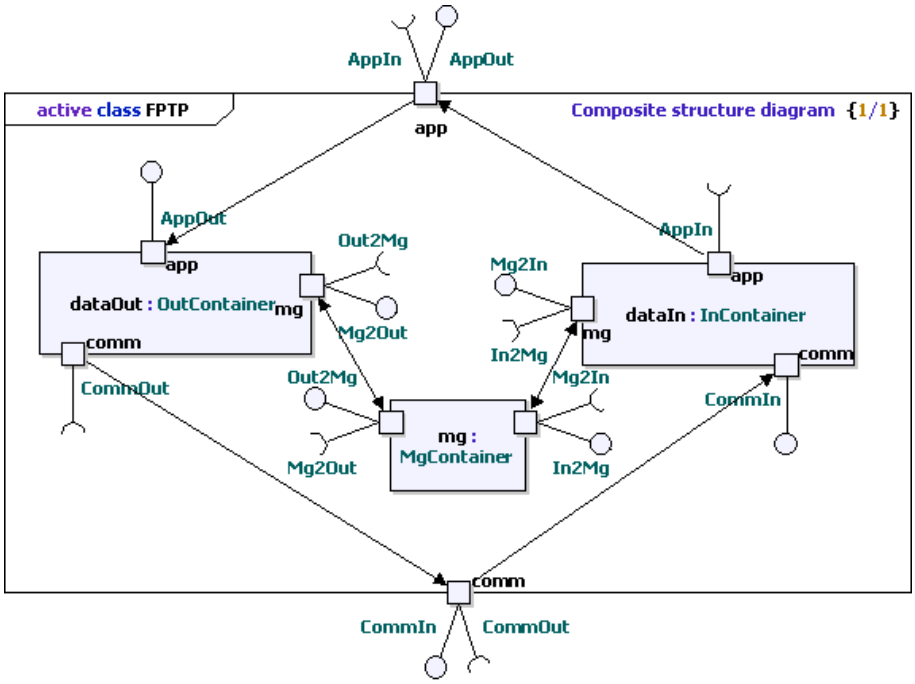

Figure 3. Transport protocol composite pattern

Note: In this document, diagrams following the OMG UML 2.3 specification will be used to represent structural and behavioral features of the transport architecture and components. These diagrams include class, object, composite, component and use case diagrams. More information can be found at http://www.omg.org/spec/UML/2.3

Figure 3 presents a legend aimed at helping reading the ontologies presented in this document. The visual representation of classes, individuals (or objects), object properties (or

relationships between individuals), data properties (or object attributes) and data values is presented. Other visual elements representing enumerations (E), unions (∪), intersections (∩), compliment (¬), universal (∀) and existential (∃) expressions could also be found the ontologies presented in this document.

This composite communication pattern is based on two kinds of containers:

- Manager container : this container is aimed at deploying 1 or several mechanisms operating at the management plane. These mechanisms are event-based (e.g. timeouts, QoS alerts, messages received from the peer entity, etc.) and communicate with the data control plane by sending and receiving management signals. An example of a manager component is an error controller mechanism able to take decisions about the retransmission of lost data if no feedback has been received after the expiration of a specific time period.
- Data input/ouput containers: these containers are intended to be used to deploy control mechanisms operating on messages being sent or received by the applications. These mechanisms communicate with the underlying communication service to send and receive data. Examples of these mechanisms are a rate control limiting the data being sent or a loss detector component able to recognize losses in the data being received. These control-level mechanisms are dynamically configured by management-level mechanisms within the manager container (e.g. by changing the sending rate or by changing the loss detection policy in order to provide a partial reliable service).

Next section presents a study about the specialized transport mechanisms provided by FPTP and shows how these mechanisms can be deployed, composed and specialized in this composite architectural pattern.

3.2. FPTP services, functions and mechanisms

In the original TCP/IP model, the two traditional protocols TPC and UDP were designed to provide either a fully reliable and a fully ordered service or a non reliable and a non ordered service. Since then, several studies have been carried out to provide new alternatives such as a partial ordered and a partial reliable (PO/PR) services in order to take advantage to the solution space comprised between TCP and UDP services.

The Partial Order Connection protocol (POC) was the first protocol proposing such PO/PR service [Amer 1994], [Diaz 1994], [Chassot 1995]. This kind of service can provide a more compliant service for time-constrained applications able to tolerate a certain degree of disorders or losses.

Several studies to apply and evaluate this family of services have been proposed. In [Connolly 1994] a TCP extension aimed at introducing PO and PR services was proposed. Likewise, studies aimed at applying and evaluating PR services for the transport of MPEG video streams [Rojas 1999] or for Reed-Solomon codes for distributed real-time MPEG

streams [Ihidoussene 2002] have been carried out. Likewise, in [Stewart 2004] a partial reliability extension to SCTP named PR-SCTP has been proposed.

FPTP protocol follows the approach proposed by POC, in order to implement several transport functions specializations based on the PR service. Moreover, FPTP offers an error-controlled, congestion-controlled and timed-controlled transport service. FPTP functions are implemented by the composition of mechanisms operating at both control and management planes [Exposito 2005]. FPTP mechanisms have been designed and developed in order to adapt the transport service to the application requirements and to the provided network service.

Figure 4 presents the several functions implemented by FPTP.

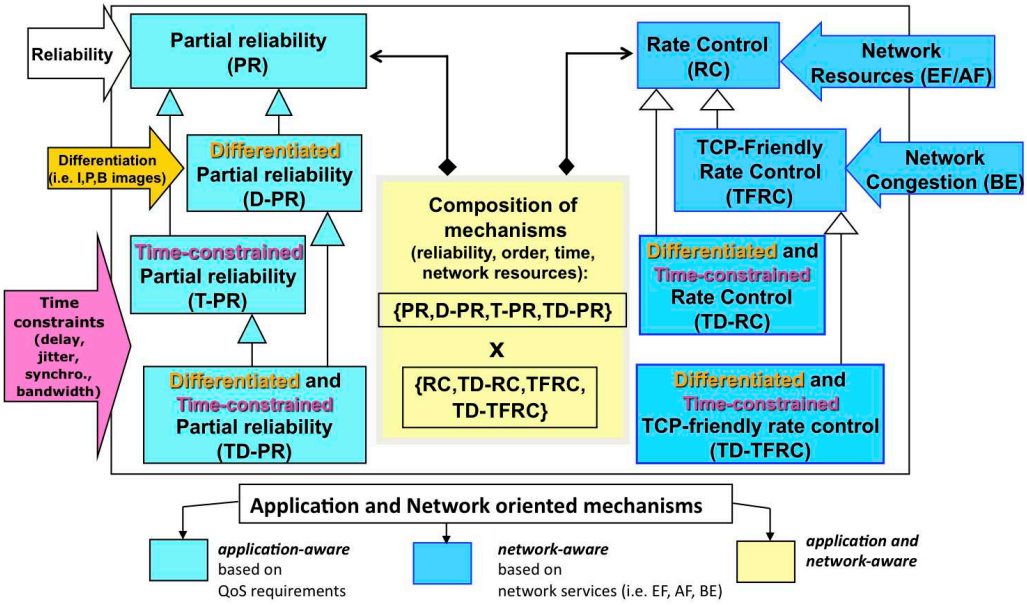

Figure 4. FPTP transport functions

FPTP functions can be classified as:

- Application-aware: from the application point of view, reliability requirements and time constraints have guided the design of application-aware functions (e.g. PR, D-PR, T-PR and TD-PR).
- Network-aware: from the network point of view, adaptation to the available throughput provided by guaranteed network services (e.g. rate control or RC) or the friendly behavior required in Best-effort networks (e.g. TCP-friendly rate control or TFRC) have guided the design of network-aware functions.
- Application and network-aware: compositions of both application-aware and network-aware functions (e.g.TD-RC, TD-TFRC, TD-PR-TD-TFRC, etc) are intended to provide a transport service adapted to the network resources while taking into account application requirements.

Figure 5 presents an use case diagram illustrating the large diversity of FPTP services implemented by these application-aware and network-aware functions.

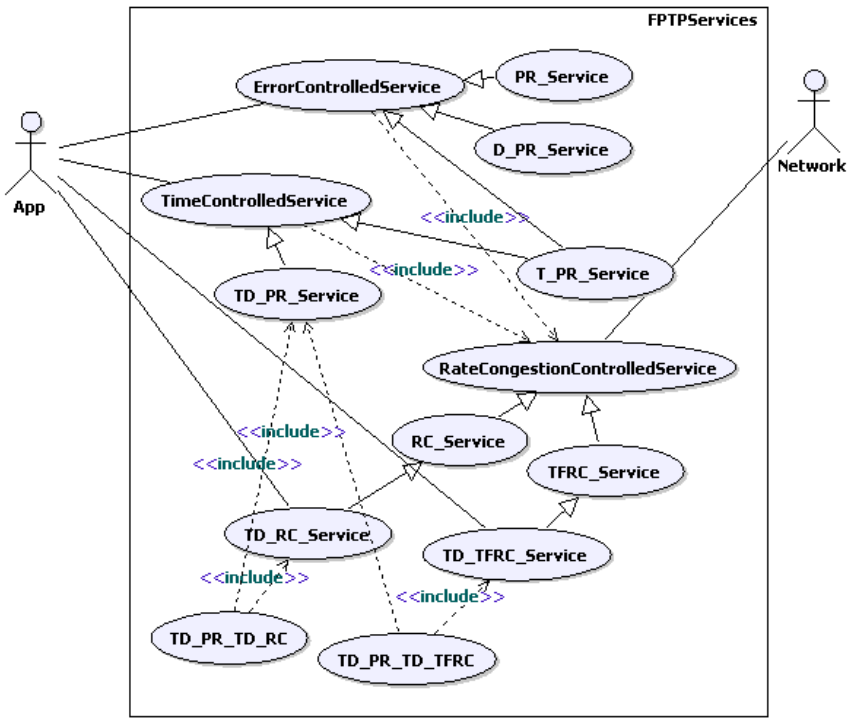

Figure 5. FPTP transport services

Figure 6 presents a class diagram illustrating the basic set of FPTP mechanisms aimed to be composed in order to implement the required transport functions.

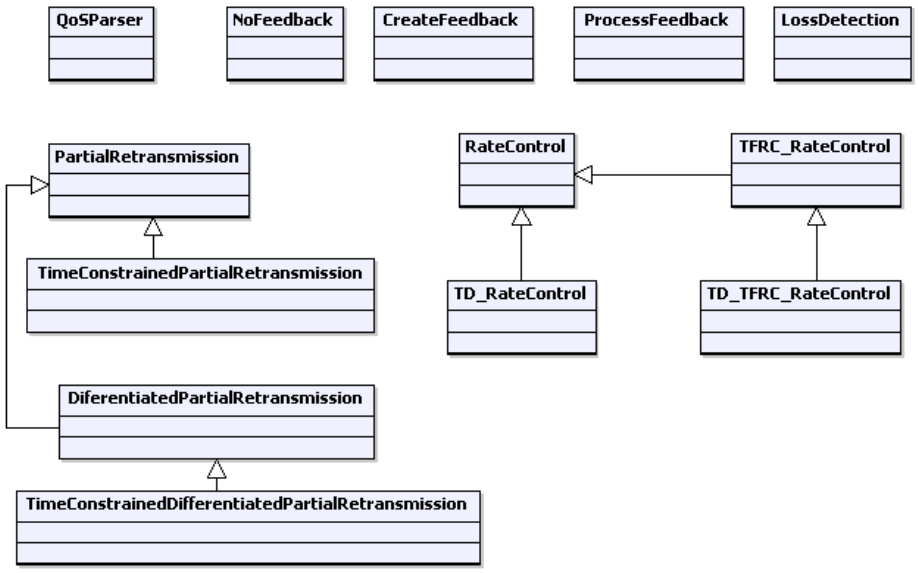

Figure 6. FPTP transport mechanisms

Next paragraphs describe the composition of these mechanisms in order to implement the FPTP transport functions.

3.2.1. Error control functions

Error-control functions are implemented by a fundamental mechanism providing a partial reliable (PR) service. The PR mechanism is based on a retransmission based scheme. Loss detection is performed at the receiving side and the loss recovery mechanism is carried out by the sender when the feedback is received or after the expiration of a retransmission timeout. The class diagram presented in Figure 7 specifies the basic components of the PR function.

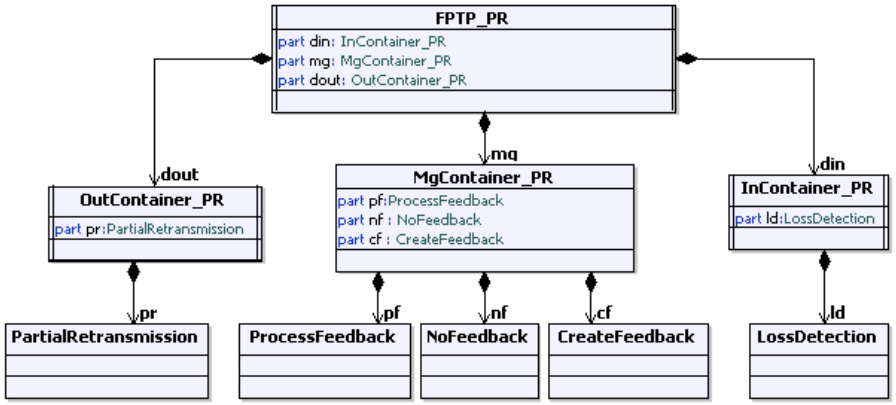

Figure 7. FPTP PR mechanism composition

The PR mechanism can be specialized in order to take into account the priorities of the different types of ADU composing the applicative stream (e.g. I, P and B pictures of MPEG video streams) [Exposito 2005b]. This differentiated and partial reliable mechanism is named D-PR (see Figure 8).

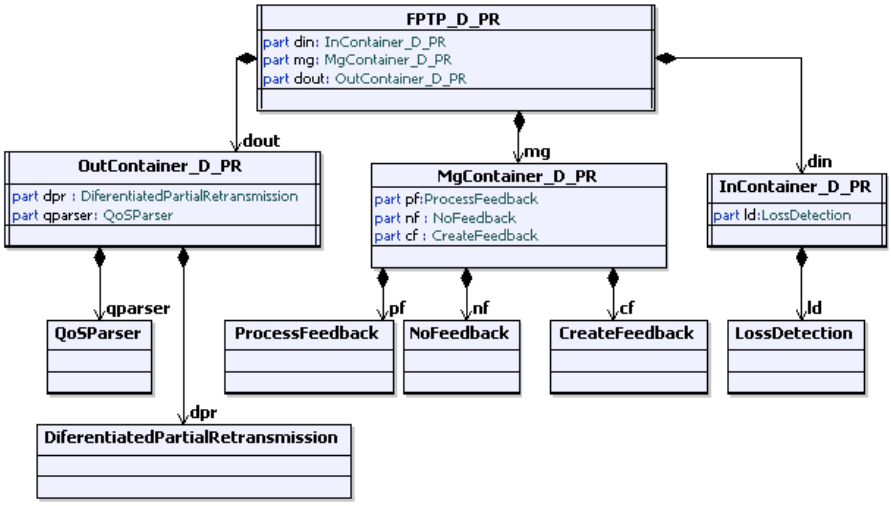

Figure 8. FPTP D-PR mechanism composition

A mechanism named QoSParser intended to perform ADU differentiation has to be deployed at the sending side. The basic ProcessFeedback mechanism included within the PR mechanism needs to be enhanced to control the partial reliable service taking into account ADU priorities. Figure 8 illustrates the components of the FPTP D-PR function. Likewise, the partial reliable mechanism can be specialized by a time-constrained partial reliable mechanism able to offer a service compliant with the reliability and time requirements. Figure 9 presents the mechanisms composed to implement the FPTP T-PR function.

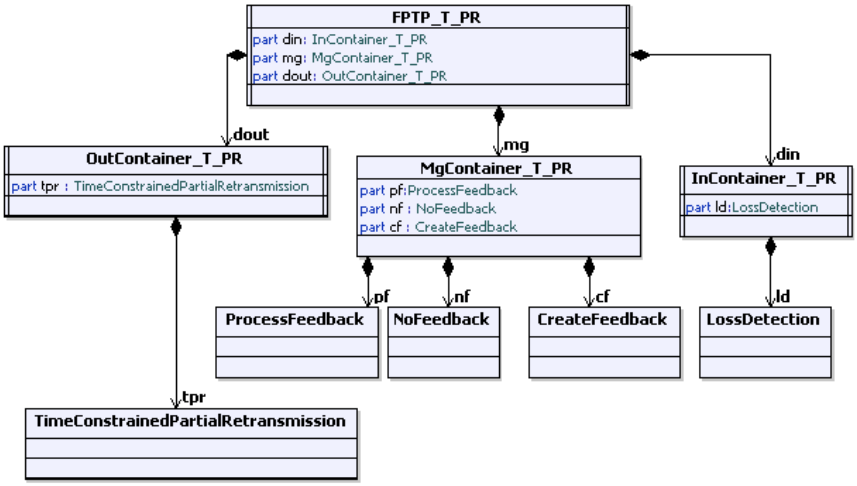

Figure 9. FPTP T-PR mechanisms composition

Similarly, in order to provide an error-controlled and time-controlled service, a new specialization named time-constrained and differentiated partial reliable mechanisms (TD-PR) has been designed. TD-PR takes into account the application time constraints in order to provide the most adapted partial reliable service. Figure 10 presents the mechanisms required to implement the FPTP TD-PR function.

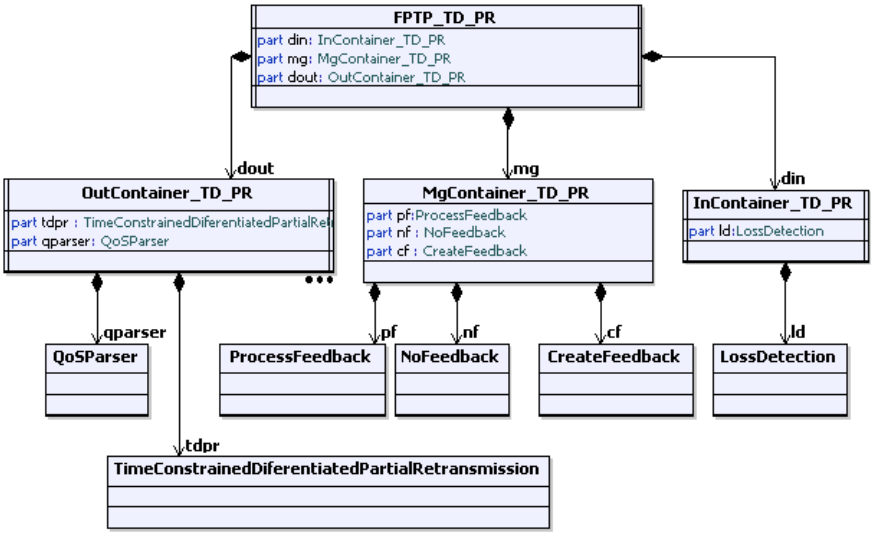

Figure 10. FPTP TD-PR mechanisms composition

3.2.2. Rate and congestion control functions

Rate control function is implemented by a basic rate control mechanism (RC) able to adapt the application data flow to the allowed sending rate. This adaptation is achieved by delaying the packets in order to respect the allowed throughput. Figure 11 presents the class diagram representing the FPTP RC function.

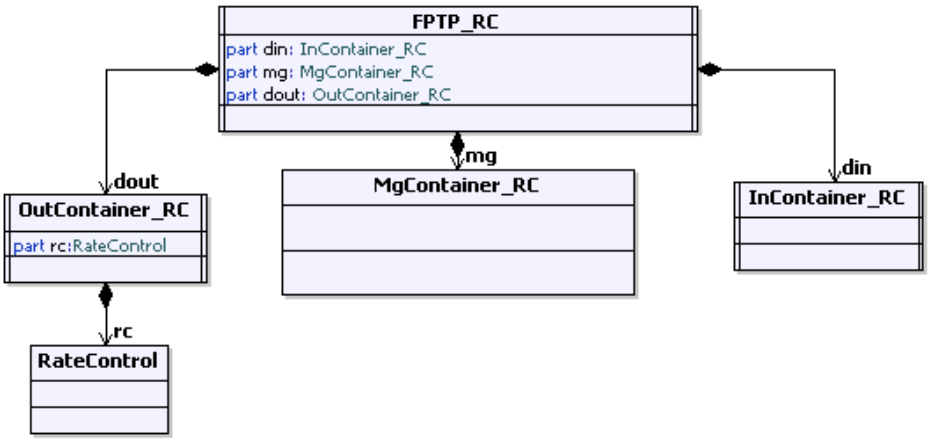

Figure 11. FPTP RC mechanism

The rate control function has been specialized in order to provide a time-constrained and differentiated rate control (TD-RC) able to take into account partial reliability tolerance and time constraints of the application data. TD-RC achieves flow adaptation by selectively discarding obsolete and less important packets if the accumulated delay generated by the rate control mechanism is not compliant with the application requirements. Figure 12 presents the class diagram representing the FPTP TD-RC function.

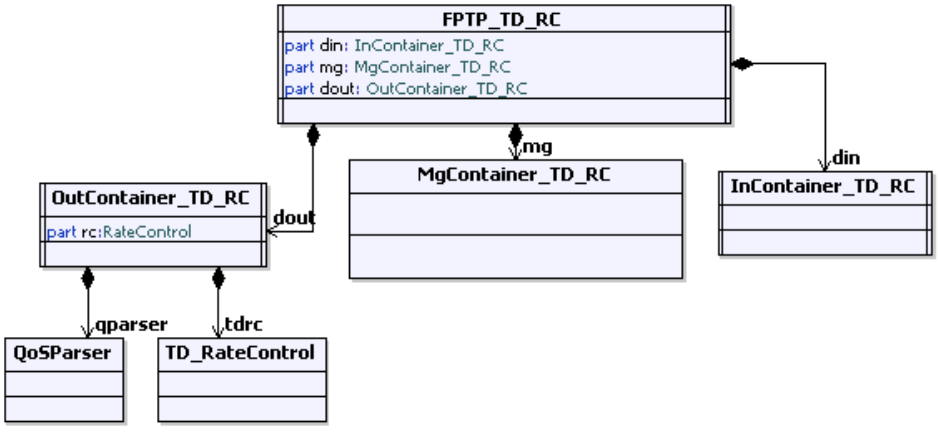

Figure 12. FPTP TD-RC mechanisms composition

The congestion control function is achieved by a composition including a specialization of the rate control mechanism named TCP-friendly rate control mechanism (TFRC). This mechanism is composed with loss detection and feedback management mechanisms in

order to compute the allowed sending rate based on the detected network conditions. Figure 13 presents the class diagram representing the FPTP TFRC function.

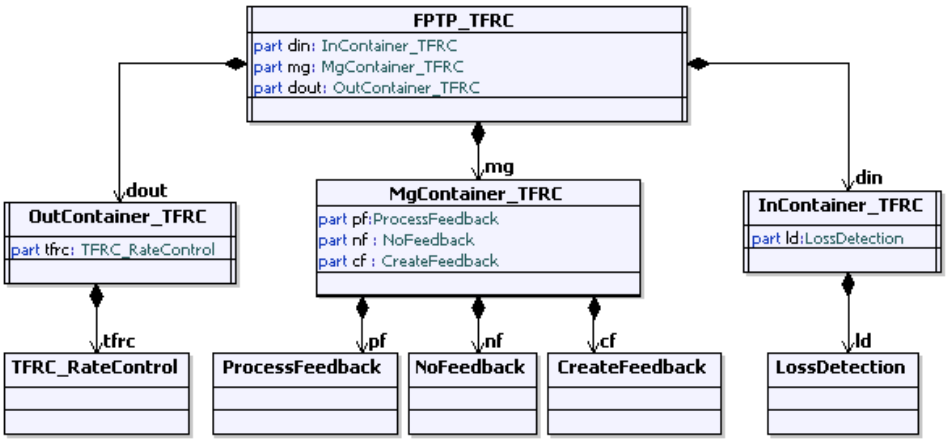

Figure 13. FPTP TFRC mechanisms composition

The TFRC mechanism has also been specialized to provide a time-constrained and differentiated TCP-friendly rate controlled function (TD-TFRC). Figure 14 presents the class diagram representing the FPTP TD-TFRC function

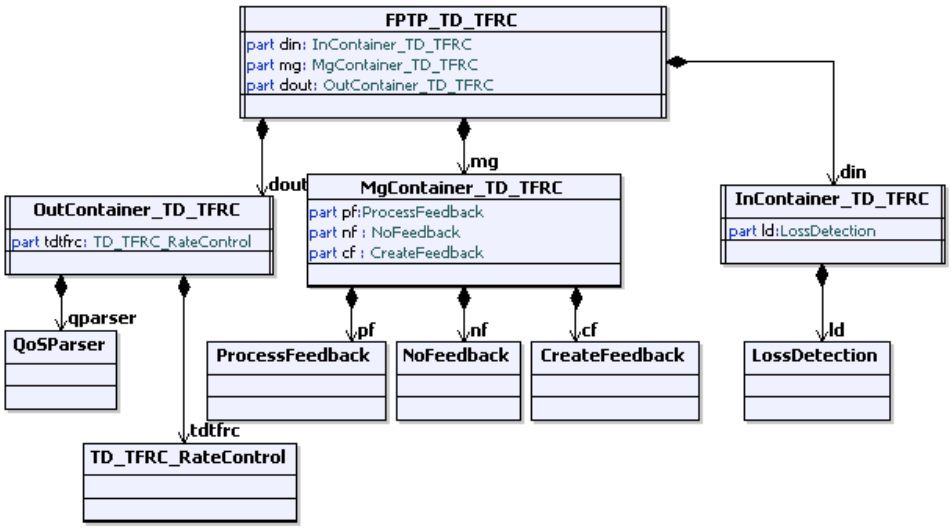

Figure 14. FPTP TD-TFRC mechanisms composition

3.2.3. Composed functions

Some of the previous functions can be composed in order to implement application-aware and network-aware functions such as error, rate/congestion and time controlled functions. For instance, the diagram presented in Figure 15 illustrates the composition of mechanisms aimed at implementing a time-constrained and differentiated partial reliable and TCP-friendly rate controlled function.

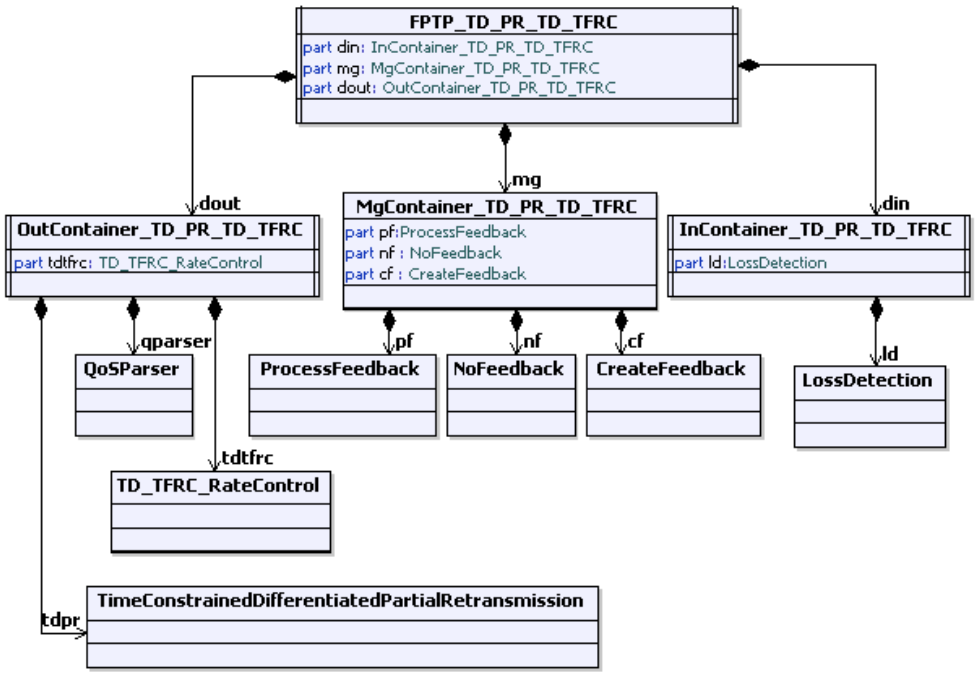

Figure 15. FPTP TD_PR + TD-TFRC mechanisms composition

Further information including the UML specification of the various mechanism provided by FPTP can be found in [Exposito 2006], [Exposito 2004], [Exposito 2003]. Likewise, the UML design specification as well as the API documentation of the FPTP protocol has been included in the annexes of this chapter.

Next section presents some experiments intended to evaluate the FPTP services.

3.3. FPTP evaluation

An important number of studies based on simulation and emulation strategies have been carried out in order to evaluate the FPTP mechanisms. We have designed and developed an UML/SDL based simulation environment in order to evaluate and validate the FPTP specification, including its behavioral and structural features [Exposito 2005a]. More information about the simulation results can be found at [Exposito 2006].

Likewise, we have carried out several experiments using an advanced network emulation platform [Dairaine 2006], [Dairaine 2006a] in order to evaluate the services offered by a FPTP java-based implementation. Next paragraphs summarize and analyze the results of a selected set of these experiments.

3.3.1. FPTP TD-TFRC mechanism

Figure 16 illustrates the evaluation of the TD-TFRC mechanism for time-constrained multimedia applications [Exposito 2003a]. Two multimedia applications producing RTP/H.263 and RTP/MPEG video flows have been evaluated.

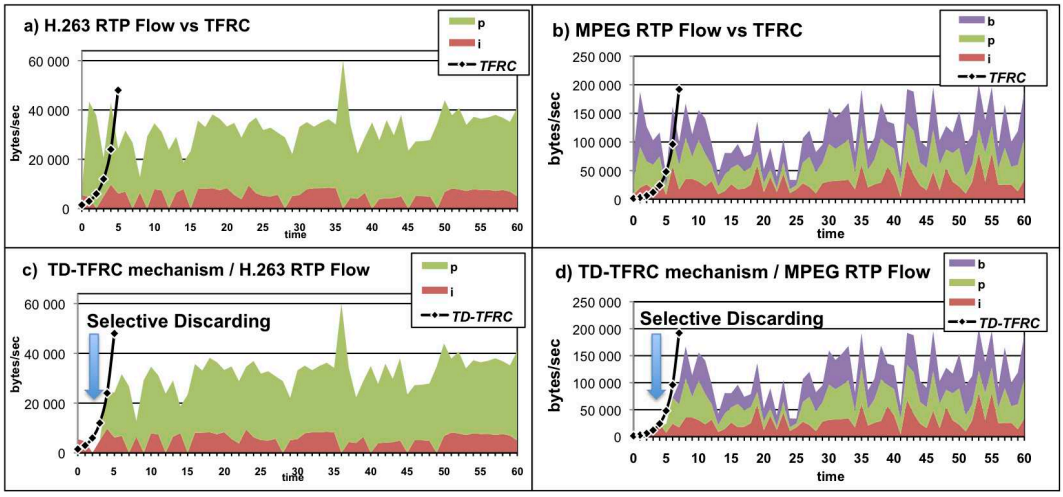

Figure 16. TD-TFRC mechanism evaluation

Figures a) and b) shows the effects of the standard TFRC mechanism over these flows where an important set of application data units should be delayed during the slow-start phase (i.e. data exceeding the sending-rate limitation imposed by TFRC). These ADU are delayed at the sending side by the standard rate control mechanism of TFRC and might arrive too late at the receiving side to be delivered. Figures c) and d) illustrate the adaptation performed by the TD-TRFC mechanism. ADU priorities and time-constraints are taken into account by TD-TFRC. TD-TFRC specializes the standard rate control mechanism by using a selective-discarding rate control mechanism. In this way, obsolete and less important ADUs will be discarded at the sending side in order to use the available sending rate to transmit more important and time-valid multimedia data.

3.3.2. FPTP D-PR and TD-PR mechanisms

Figure 17 shows the evaluation of the TD-PR and D-PR mechanisms for time-constrained multimedia applications [Exposito 2005]. A multimedia application producing RTP/H.263 video flows has been evaluated over a lossy emulated network environment.

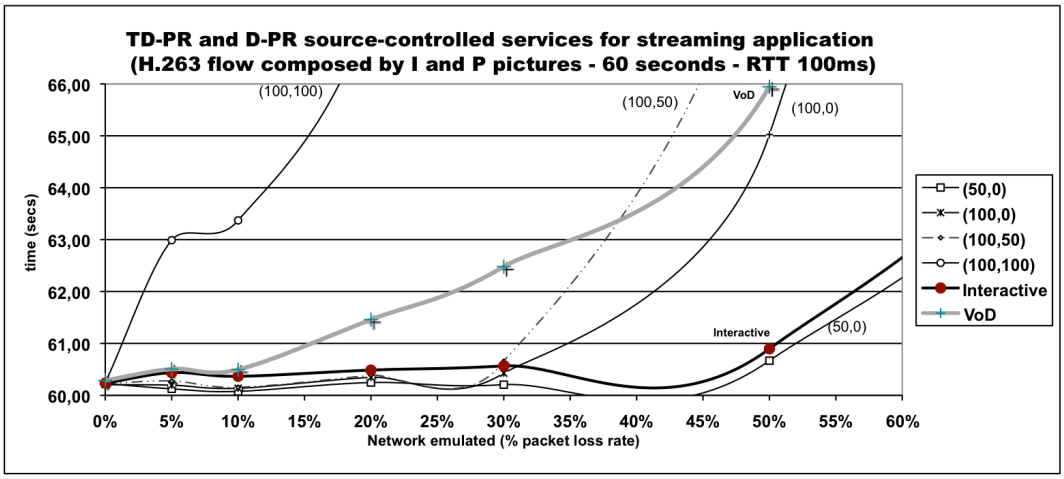

Figure 17. TD-PR and D-PR mechanisms evaluation

The D-PR mechanism has been evaluated using 4 different configurations of the expected partially reliable service. These configurations are defined in terms of the differentiated partial reliability to be guaranteed for the transmission of more important (i.e. I pictures) and less important (i.e. P pictures) ADUs composing the video flow. These 4 configurations are the following:

- D-PR1 = {(I,P)=(50%,0%)}
- D-PR2 = {(I,P)=(100%,0%)}
- D-PR3 = {(I,P)=(100%,50%)}
- D-PR4 = {(I,P)=(100%,100%)}; note that D-PR4 is equivalent to FR.

Likewise, two configurations of the TD-PR mechanisms have also been evaluated. These configurations are intended to offer services compliant with interactive and Video on Demand (VoD) applications:

- For the interactive application, TD-PR is configured to offer the best service comprised between D-PR1 and D-PR4 while respecting an end-to-end delay lower than 400ms.
- For the VoD application, TD-PR is configured to offer the best service comprised between D-PR1 and D-PR4 while respecting an end-to-end delay lower than 10 seconds.

These experiments have permitted to evaluate the gains in terms of delay reduction by accepting a partially reliable service as the one offered by FPTP. Results show that the various configurations of D-PR and TD-PR have satisfied the applications requirements in terms of reliability. Likewise, the advantages of using a D-PR service for media flows with differentiated reliability requirements have been demonstrated. Moreover, these results show that TD-PR is able to offer the highest reliable service while respecting the time constraints.

3.4. Conclusions and lessons learned

In this section, the design and development of an error-controlled, congestion-controlled and time-controlled transport protocol has been presented. Various mechanisms and composition of mechanisms required to provide the adequate transport functions have been presented and evaluated. The compositional architecture promoted by FPTP facilitates the specialization and development of new mechanisms aimed at increasing the diversity of offered transport services. However, an efficient approach guiding service discovery, composition and deployment is required in order to facilitate the use of the current and future generation of this kind of transport services. Indeed, the specializations and the implementations of new mechanisms needs a specialized architecture aimed at facilitating publication and discovery of new services in order to allow current and future applications to bind to the most adapted transport services. We believe that solutions in the area of service-oriented and service-component architectures could satisfy these requirements. Next section provides some guidelines and perspectives on using these paradigms within a semantic-driven approach for the design of the next generation transport layer.

4. State of the art on software architectural frameworks

The complexity involved and the constant evolution of the large diversity of transport protocols and mechanisms raise the needs for new approaches aimed at supporting dynamic discovery and deployment of services based on application requirements and network constraints. Service-oriented and service-component architecture approaches offer potential solutions able to answer to these needs.

4.1. Service-Oriented Architecture

The architecture of a software system is a specification of the fundamental organization of its components, their relationships to each other and to the environment and the principles guiding its design and evolution [IEEE-1471 2000].
A Service-Oriented Architecture (SOA) is an architectural framework or referential model for building software systems based on distributed services which may be offered by different service providers [SOA RM 2006].

In [SOA ML 2009], SOA is defined as an architectural paradigm for defining how people, organizations and systems provide and use services in an agile, scalable and interoperable world.

SOA software architectures are based on the following key concepts [Krafzig 2004]:

- Service participants:
  - Service consumer: entity making use of the service offered by a service producer. A consumer looks up a service repository and identifies the details about the service including its interface. Once the service has been located, the consumer invokes it using the appropriate mechanism.
  - Service producer: entity offering a specific service or functionality. A producer usually registers the functionality that it provides and the interface that has to be invoked to make use of the service in a service repository.

- Services:
  - Service contract: formal or informal specification of the service including the purpose, functionalities, constraints and usage of the service.
  - Service interface: specifies how the service can be accessed, including data format and operations. It also identifies the invocation mechanism to invoke the service.
  - Service implementation: a physical implementation of the service providing the required capability or business logic.
- Service repository: provides the required facilities to discover and use services. A repository stores the details about the services that can be invoked (i.e. service contract) and how to invoke them (i.e. service interfaces, physical location, etc.)
- Service bus: provides the required connectivity between the service participants. A service bus may be able to connect participants using heterogenous technologies or environments.

In the SOA framework, a service is accessed using the service interface and the resulting service has to be consistent with the service description or contract. A service producer exposes its functionalities in the form of services that can be reused across different applications. Service consumers are loosely coupled to the service producers and can bind to the service at development-time or runtime.

Service consumers can bind to the services provided by the service producer at development-time or runtime:

- Development-time binding: the developer is responsible for locating all required information from the service repository in order to program the service consumer to access the service.
- Runtime binding: the service is dynamically bound based on its name, its properties or using reflection.
  - Runtime name-based binding: the service name and interface is known at development time and the service consumer can be statically programmed.
  - Runtime properties-based binding: one or several services interfaces are known at development time and the adequate service is discovered at runtime based on its properties.
  - Runtime reflection-based binding: the service interface is not known at development time and the consumer needs to dynamically interpret the semantics of the service.

Figure 18 presents a semantic model integrating the various SOA entities, concepts and relationships.

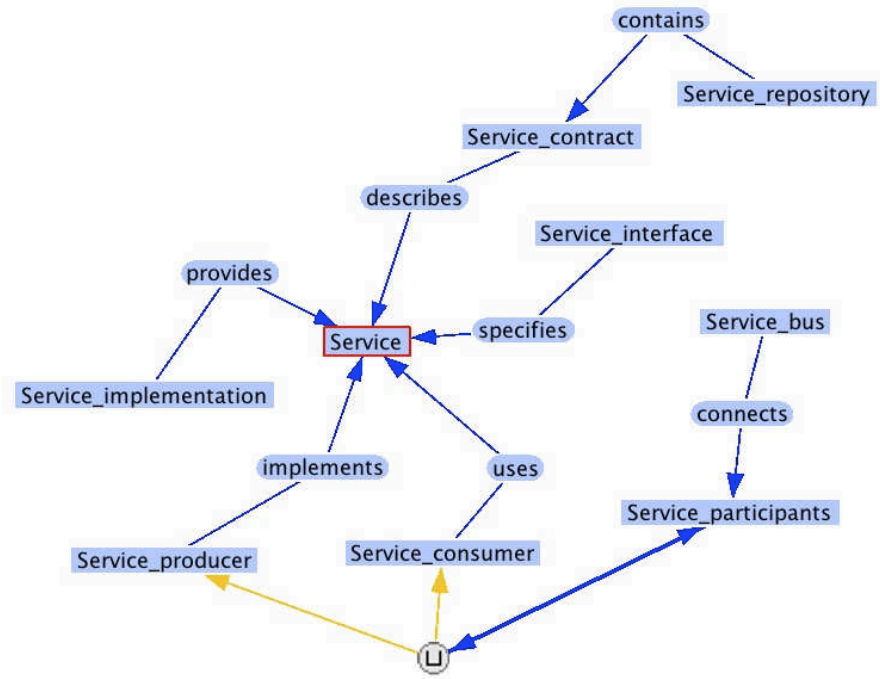

Figure 18. SOA semantic model

4.2. Component-Based Design

Service Component Architecture (SCA) is a standard proposed by the OASIS consortium for composing and deploying service-oriented systems. The SCA framework provides an assembly model for building systems based on Service Oriented Architectures (SOA) and based on composites and service components [SCA AssemblyModel 2007].

In this model, an application or system results of the assembly of interconnected service components. The composition of service components allows implementing service-oriented systems that could be distributed. Moreover, these systems can be composed of heterogeneous components (e.g. PHP, JavaScript, java, C/C++, web-services or BPEL implementations).

Service components are the basic elements of a composite system. A service component consists of a configured instance of a component implementation providing a specific functionality. Components can publish and share properties. A component may depend on services provided by other components. Components publish or implement “service” interfaces and can require or use “reference” interfaces. Interfaces are “wired” when the referenced interface is connected to an implemented interface. Interfaces are promoted when the service is published. A service implementation is configured when appropriate values are given to the properties and when the referenced interfaces are wired to the required services. The components can be "wired" locally in the same system or remotely using an adequate protocol bindings (e.g. Corba IIOP, web services, http, etc.). A composite contains components, services, references, property declarations, plus the wiring that describes the connections between these elements. A set of interrelated composites within the same vendor's SCA implementation forms a Domain.

The Service Component Definition Language (SCDL) is an XML-based language aimed at defining all the elements of a SCA composite. SCDL allows characterizing components and composites as well as specifying the relationships between them. SCDL works like a deployment descriptor for SCA applications. Figure 19 illustrates the SCA assembly model.

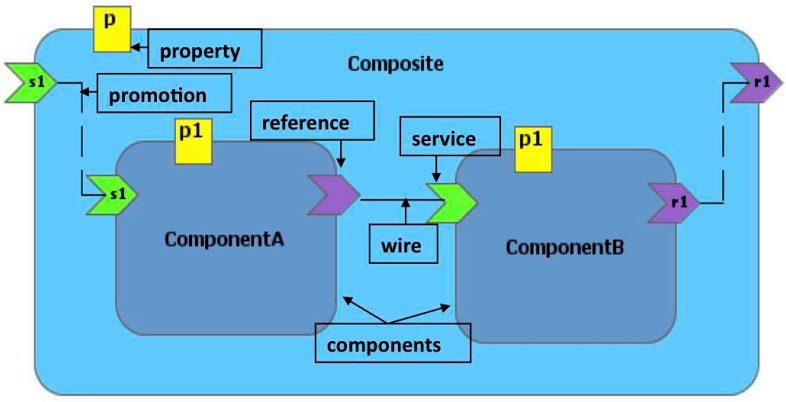

Figure 19. Service Component Architecture

Figure 20 presents a semantic model integrating the SCA entities and definitions introduced in this section.

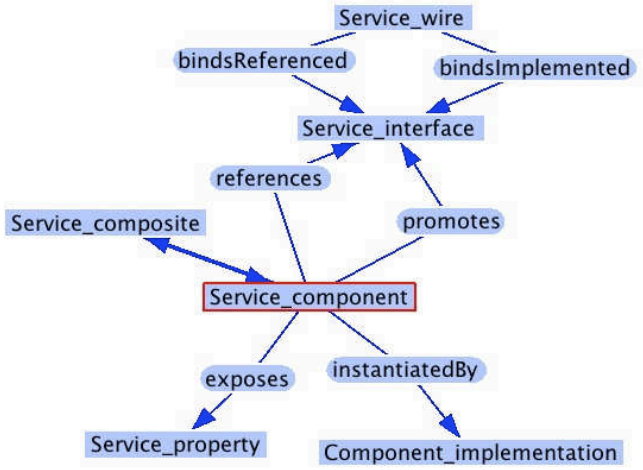

Figure 20. Service Component Architecture semantic model

4.3. Summary

Service-oriented and service-component architecture approaches presented in this section offer potential solutions for dynamic discovery, selection, composition and deployment of transport mechanisms and functions. Next section presents several guidelines aimed at applying these approaches enriched by semantic models in order to design the next generation transport layer.

5. Design guidelines of a component-based and service-oriented architecture for the next generation transport layer

Based on the transport architecture framework proposed by FPTP, the next generation transport layer should follow the component-based and the service-oriented architectural approaches guided by a common semantic model in order to allow dynamic configuring based on the application requirements and on the available network context. This section presents several guidelines and perspectives aimed at guiding the design of the next generation transport layer.

5.1. Service-oriented architecture transport layer (SOATL)

A service oriented architecture transport layer (SOATL) is intended to cope with the complexity involved in discovering and selecting the adequate service among the large diversity of current transport services. The following are the functional requirements guiding the SOATL design:

- The SOATL should provide a transport services repository accessible from a transport service bus where application programmers would be able to discover the available services.
- During development-time, the service interfaces retrieved from the SOATL will be used to program the applications in order to bind to the adequate service.
- During runtime, the most adequate service implementation will be used based on the current network context. Indeed, several transport service implementations could provide the same service contract even if some implementations could be more adapted to a specific network context.
- Furthermore, in order to provide a more flexible and open architecture, both service discovery and binding could be done during runtime based on the service properties deduced from the application requirements. This approach will allow a more efficient and smoother integration of future transport services without requiring applications upgrading.

5.2. Service-component architecture for transport protocols (SCATP)

SCA model proposes an adapted framework that can be used to design and develop composite transport protocols such as FPTP. The following are some design requirements of the SCATP architecture.

- A generic service-component architecture for transport protocols (SCATP) should allow dynamically composing the adequate transport functions guided by the services required by the applications.
- SCATP should provide dynamic composition of elementary transport mechanisms (e.g. error detection, error recovery, etc) intended to implement the adequate transport functions (e.g. error control, congestion control etc).
- The resulting dynamic composite transport protocol architecture would consist in a service composite containing a set of pluggable transport mechanisms following the SCA assembly model adapted to the specificities of end-to-end transport protocols. These specificities include integrating data/control and management planes for mechanisms deployment at sending and receiving sides.

5.3. Semantic model guiding the selection and composition of transport services

Dynamicity, extensibility and heterogeneity properties required by SOATL and SCATP architectures can be guaranteed by a semantic model aimed at guiding service selection and composition.

This semantic model has to be based on assertions about the mechanisms implemented by each transport protocol. These assertions are intended to be processed by reasoning technologies in order to infer the characterization of services offered by the transport protocols.

Likewise, the semantic model should contain assertions about the mechanisms that can be composed by configurable transport protocols such as FPTP. These assertions are intended to infer the required mechanisms aimed at delivering specific composite transport services.

The QoS transport ontology model designed in the previous chapter has been enhanced in order to integrate the required assertions intended to describe the mechanisms implemented by standard protocols and the compositions of mechanisms provided by components-based protocols following the SCATP approach. Likewise, based on the SOATL approach, assertions defining transport services characterization that can be offered has been incorporated in this ontology.

Figure 21 illustrates how service characterizations based on the assertions describing mechanisms composition of the FPTP protocol are inferred by a standard reasoner (i.e. Pellet) using the QoS transport ontology.

Figure 21. Error, throughput and time controlled service characterization

- The error, throughput and time controlled service class is defined as the intersection of error controlled, throughput controlled and time controlled services.
- The reasoner engine infers that FPTP is the only protocol instance providing this class of service.
- The axioms causing this inference are based on the service class definition and on the mechanisms implemented by FPTP (e.g. FPTP_FR provides an error controlled service, FPTP_RC provides a throughput controlled service and FPTP_TD_PR provides and time controlled service).

Similarly, assertions describing the mechanisms and functions implemented by standard transport protocols have been incorporated in the QoS transport ontology presented in chapter 1. This new version of the transport ontology is aimed at being incorporated within the SOATL architecture to be used by application programmers in order to allow dynamic discovery and selection of transport services. Likewise, this semantic model also incorporates descriptions about fine grained transport mechanisms and their possible compositions provided by transport protocol programmers. This component-based semantic information is aimed at guiding the mechanisms composition process (discovery and selection) to be guaranteed by the SCATP architecture.

Further information about the classes, individuals and properties definitions of this ontology can be found in the annexes section. Likewise, the semantic description of standard (TCP, UDP, SCTP, DCCP and MPTCP) and component-based (FPTP) protocols including the assertions about their mechanisms and the inferences characterizing their services have been included in the annexes of this document.

6. Conclusions

In this chapter the design of a service-oriented and component-based architecture aimed at integrating existing and new generation of transport services has been presented. An ontology-driven approach has been followed in order to provide service-oriented discovery and selection of transport services based on the semantic transport services characterization. Likewise, transport components semantic suited for compositional transport services has also been presented.

The architecture proposed by SOATL facilitates the integration of traditional and new transport protocols intended to satisfy the application requirements while dealing with new network services and technologies. Likewise, this framework is well suited to integrate component-based protocols such as FPTP. The service-component architecture approach promoted by SCATP offers an assembly model well suited for pluggable transport components composition and deployment guided by the semantic associated to the required transport service. SOATL and SCATP architectures and the semantic model guiding their operations allow implementing the required dynamic configuration functionalities for the transport layer of the next generation.

Based on this semantic-driven approach, further studies aimed at enhancing this architecture in order to provide adaptation capabilities required to cope with dynamic network environments need to be carried out. The autonomic computing framework provides a well suited architectural framework able to integrate both the self-configuring and the self-adapting functionalities of an autonomic transport layer.

7. Annexes

7.1. FPTP

- The FPTP UML specification can be found at http://homepages.laas.fr/eexposit/hdr/fptp/uml/
- The documentation of the FPTP java implementation can be found at http://homepages.laas.fr/eexposit/hdr/fptp/javadoc
- The FPTP java implementation can be found at http://homepages.laas.fr/eexposit/hdr/fptp/FPTP.jar
- The FPTP UML specification in XMI format is available at: http://homepages.laas.fr/eexposit/hdr/fptp/xmi/FPTP.xml (version txt:http://homepages.laas.fr/eexposit/hdr/fptp/xmi/FPTP.xml)

7.2. Ontologies

- The SOA ontologies can be found at http://homepages.laas.fr/eexposit/hdr/SOA.owl (version txt at http://homepages.laas.fr/eexposit/hdr/SOA.txt)
- The SCA ontologies can be found at http://homepages.laas.fr/eexposit/hdr/SCA.owl (version txt at http://homepages.laas.fr/eexposit/hdr/SCA.txt)
- The QoS Transport ontology defined in this study is available at http://homepages.laas.fr/eexposit/hdr/QoSOntology_V2.owl (version txt at http://homepages.laas.fr/eexposit/hdr/QoSOntology_V2.txt)

7.3. Semantic description of standard transport protocols

- TCP semantic description: http://homepages.laas.fr/eexposit/hdr/tcp/
- UDP semantic description: http://homepages.laas.fr/eexposit/hdr/ucp/
- SCTP semantic description: http://homepages.laas.fr/eexposit/hdr/sctp/
- DCCP semantic description: http://homepages.laas.fr/eexposit/hdr/dccp/
- MPTCP semantic description: http://homepages.laas.fr/eexposit/hdr/mptcp/
- FPTP semantic description: http://homepages.laas.fr/eexposit/hdr/fptp/

8. References

[Amer 1994] Paul Amer, Christophe Chassot, C. Connolly, P. Conrad, Michel Diaz, Partial Order Transport Service for Multimedia and other Applications, IEEE ACM Transactions on Networking, 2, 5, October, 1994

[Chassot 1995] Christophe Chassot, Architecture de Transport Multimédia à Connexions d'Ordre Partiel, Institut National Polytechnique de Toulouse, December, 1995

[Connolly 1994] P. Conrad, P. Amer, T. Connolly, An Extension to TCP : Partial Order Service, IETF RFC 1693, November, 1994

[Dairaine 2006a] Laurent Dairaine, Ernesto Exposito, Hervé Thalmensy, Towards an unified experimentation framework for protocol engineering, 1st International Workshop on Service Oriented Architecture in Converging Networked Environments, Vienna (Austria), 2, 18-20 april, 2006

[Dairaine 2006] Laurent Dairaine, Ernesto Exposito, Guillaume Jourjon, Pierre Casenove, Feiselia Tan, Emmanuel Lochin, IREEL : Remote Experimentation with Real Protocols and Applications over Emulated Network, poster in proceedings of ACM/ITiCSE'06 (Eleventh Annual Conference on Innovation and Technology in Computer Science Education), Bologna, Italy, 26-28 June, 2006

[Diaz 1994] Paul Amer, Christophe Chassot, André Lozes, Michel Diaz, Partial Order Connections: A new concept for High Speed and Multimedia Services and Protocols, Annals of Telecommunications, 49, 5-6, May-June, 1994

[Exposito 2003] Ernesto Exposito, Specification and implementation of a QoS oriented transport protocol for multimedia applications, Ph.D. dissertation, Université de Toulouse, Institut National Polytechnique de Toulouse, December 17, 2003

[Exposito 2003a] Ernesto Exposito, Patrick Senac, Michel Diaz, FPTP: the XQoS aware and fully programmable transport protocol, 11th IEEE International Conference on Networks (ICON'2003), Sydney (Australia), 28 September - 1st October, 2003

[Exposito 2004a] Ernesto Exposito, Patrick Senac, Michel Diaz, UML-SDL modelling of the FPTP QoS oriented transport protocol, 10th International Multimedia Modelling Conference (MMM'2004), Brisbane (Australie), 5-7 Janvier, 2004

[Exposito 2004] Ernesto Exposito, Michel Diaz, Patrick Senac, Design principles of a QoS-oriented transport protocol, IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM'2004), Bangkok (Thaïlande), 23-26 Novembre, 2004

[Exposito 2005] Ernesto Exposito, Patrick Senac, Michel Diaz, Compositional Architecture Pattern for QoS-Oriented Communication Mechanisms, 11th International Multimedia Modelling Conference (MMM'05), Melbourne (Australia), January 12-14, 2005

[Exposito 2005a] Ernesto Exposito, Robert A. Malaney, Xun Wei, Nghia T. Dao, Using the XQoS Platform for designing and developing the QoS-Seeker System, 3rd International IEEE Conference on Industrial Informatics (INDIN'05),track Service Oriented Architectures in Converging Networked, Perth (Australia), 10-12 August, 2005

[Exposito 2005b] Ernesto Exposito, Laurent Dairaine, Mathieu Gineste, The QoSxLabel: a quality of service cross-layer label, 3rd International IEEE Conference on Industrial Informatics (INDIN'05),track Service Oriented Architectures in Converging Networked, Perth (Australia), 10-12 August, 2005

[Exposito 2006] Ernesto Exposito, Nicolas Van Wambeke, Christophe Chassot, Michel Diaz, UML 2.0 based methodology for designing and developing real-time and QoS-oriented communication protocols, International Congress ANIPLA 2006, Rome (Italy), 13-15 November, 2006

[Hutchinson 1991] Hutchinson N., Peterson L., The x-kernel: An architecture for implementing network protocols, IEEE Transactions Software Engineering, 17, 1, 1991

[IEEE-1471 2000] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems IEEE Standard 1471-2000, September, 2000

[Ihidoussene 2002] D. Ihidoussene D, V. Lecuire, F. Lepage, Un transport à fiabilité partielle par codes de Reed Solomon pour les flux vidéo MPEG temps réels, 9ème Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'02), Montréal, May 27-30, 2002

[Krafzig 2004] Dirk Krafzig, Karl Banke, Dirk Slama, Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall, November 09, 2004

[Ritchie 1984] Ritchie D.M., A stream input-output system, AT&T Bell Laboratories Technical Journal, 63, 8, 1984

[Rojas 1999] Michel Diaz, Patrick Senac, Laurent Dairaine, Emmanuel Chaput, Luis Rojas, Video Transport Over Partial Order Connections, Computer Networks, 31, 7, April, 1999

[SCA AssemblyModel 2007] SCA Service Component Architecture, Assembly Model Specification, OASIS, March, 2007

[SOA ML 2009] Service oriented architecture Modeling Language (SoaML), OMG Adopted Specification,

[SOA RM 2006] Reference Model for Service Oriented Architecture 1.0, OASIS Standard, October, 2006

[Schmidt 1993] D. Schmidt, D. Box, T. Suda, ADAPTIVE: A Dynamically Assembled Protocol Transformation, Integration, and eValuation Environment, Journal of Concurrency: Practice and Experience, 5, 1, 1993,

[Stewart 2004] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, P. Conrad, SCTP Partial Reliability Extension, RFC 3758, May, 2004

[Wong 2001] R. Schlichting, M. Hiltunen, G. Wong, A configurable and extensible transport protocol, INFOCOM 2001, Anchorage, AK (USA), April, 2001

Chapter III: Autonomic Transport Layer

1. Introduction

Certainly, the main service offered by the transport layer is the adaptation of the network services to the application requirements. Transport protocols are intended to offer an adapted service by implementing the adequate mechanisms able to cope with the variations of the network conditions while still taking into account application requirements and preferences. When the network layer does not offer the required guarantees in terms of bandwidth, reliability or delay, common transport protocols have a limited room to maneuver. However, more efficient adaptive transport mechanisms could be designed if enough information about application preferences, requirements and constraints is available at the transport layer. These adaptive mechanisms would efficiently operate by continuously adapting to the limited or abundant dynamic resources of network environments.

This chapter introduces our work focused on designing the Enhanced fully programmable Transport Protocol (ETP). ETP results from the specialization of the FPTP component-based transport protocol presented in the chapter 2. ETP has been designed to provide an adaptive-oriented architecture facilitating dynamic behavioral and structural adaptation based on the perceived network conditions. Firstly, behavioral adaptation consists in dynamically tuning transport mechanisms in order to be more efficient face to advantageous or disadvantageous network conditions. This kind of adaptation can be achieved if enough information of application data characterization is available at transport layer. For this reason, ETP integrates an interpreter of the QoS application-layer semantic intended to be used to design adaptive transport mechanisms able to efficiently implement behavioral adaptation. This interpreter has been built based on a semantic model able to express the implicit constraints of RTP-based streaming multimedia applications. Secondly, structural adaptation can be achieved by replacing one or several of the deployed transport mechanisms with more efficient implementations based on the observed network conditions. Structural adaptation approach usually represents a more expensive solution than behavioral adaptation due to the needs of defining transitional stages while deploying and undeploying transport mechanisms. For this reason, behavioral adaptation should be performed before structural adaptation. ETP also integrates a model-driven approach aimed at guiding both behavioral and structural adaptation of transport mechanisms.

In the previous chapter, the needs for a service-oriented, component-based and semantic-driven architecture intended to design and develop transport services able to satisfy a large diversity of application requirements over heterogeneous network technologies have been presented. In this chapter, the autonomic computing paradigm will be studied in order to define and implement the adequate strategies required to dynamically adapt transport mechanisms to the changes observed in the network context. This paradigm is well suited to self-manage adaptive components based on the monitored environment conditions and guided by the service user goals. Based on this paradigm, we will present

The ETP protocol can be classified as an adaptive transport protocol according to the autonomic maturity process described in [Kephart 2003]. Following the classical autonomic

maturity process, we have initiated several works aimed at enhancing the adaptive ETP protocol in order to design and implement an Autonomic Transport Protocol (ATP). ATP should provide an autonomic architecture able to integrate knowledge base, policy orientation as well as self-managing functionalities. Behavioral and structural adaptation strategies promoted by ETP should be redesigned following the autonomic computing paradigm in order to allow self-managing and orchestration functionalities. Likewise, our initial studies aimed at designing an Autonomic Transport Layer (ATL) able to select, compose, deploy and orchestrate transport protocols following the autonomic computing paradigm and integrating a semantic-driven, service-component, service-oriented approache will be presented.

This chapter is structured as following. Section 1 introduces the Enhanced Transport Protocol (ETP), its adaptive architecture and the strategies promoted for implementing structural and behavioral adaptation. Section 2 presents a state of the art on autonomic computing paradigm including the promoted architecture, the basic self-managment functions as well as the autonomic knowledge-based and policy-oriented approach. Based on the advantages and limitations of the ETP protocol, section 3 presents several guidelines and perspectives in applying this paradigm in designing the autonomic transport protocol (ATP) as well as the autonomic transport layer (ATL) of next generation. Finally several conclusions and perspectives are presented.

2. The Enhanced Transport Protocol

2.1. Introduction

Based on the lessons learned from the design and development of the component-based FPTP protocol, we have carried additional studies in order to design and develop a new architecture and new mechanisms aimed at providing an adaptive Enhanced Transport Protocol (ETP). FPTP architecture promotes the design and composition of a large diversity of transport mechanisms and functions based on network constraints and application requirements. However, the selection, composition and deployment of these mechanisms are performed during the session establishment phase, based on the initially observed environmental context. During the transmission phase, changes on network conditions might generate positive or negative effects on the performance achieved by the provided transport service. As a consequence, the initial selection of the mechanisms (e.g. ARQ-based or FEC-based error control mechanism) and their initial configuration (e.g. partial reliability tolerance of X %), could be improved by an additional runtime adaptation based on the perceived network environment.

In ETP, transport service adaptation can be achieved following a behavioral or a structural approach [Van Wambeke 2008]. Behavioral adaptation can be achieved by using adaptive algorithms able to implement more efficient strategies based on the current environment state (e.g. by tuning mechanisms parameters such as partial reliability). Structural adaptation can be achieved by replacing one or several of the deployed transport mechanisms with

more efficient implementations. Generally, the structural adaptation approach leads to a more expensive solution than behavioral adaptation. Indeed, structural adaptation requires defining an efficient workflow comprising an initial phase for the deployment of new components followed by a transitional phase where both configurations are enabled (thus avoiding service interruption) and ending by a final phase where the old mechanisms are undeployed. For this reason, behavioral adaptation is usually performed before structural adaptation.

In this section, the specification of the adaptive communication architecture provided by ETP is firstly presented [Van Wambeke 2007], [Van Wambeke 2008b], [Exposito 2008a]. Secondly, a semantic model representing QoS application layer preferences, requirements and constraints and aimed at designing behavioral-adaptive transport mechanisms is presented [Exposito 2008], [Exposito 2009], [Gineste 2009]. Finally, a model-driven approach aimed at introducing the design of strategies for structural adaptation of the transport mechanisms in order to dynamically adapt the transport service to the available network conditions is presented [Chassot 2006], [Armando 2007], [Van Wambeke 2008a], [Guennoun 2008], [Van Wambeke 2008c].

2.2. Adaptive composite communication architecture

The communication pattern provided by FPTP allows the deployment of control and management plane components intended to implement standard and specialized transport protocol mechanisms such as error control, rate control, congestion control or time control. These basic mechanisms are designed to operate by their own, based on their view of the external environment and targeting their own objectives. It means that the environment conditions are measured (e.g. delay, losses, congestion, etc.) and the management plane components take the adequate management decisions based on the requirements expressed at deployment time.

However, when the context evolves, the policies implemented by these mechanisms could be less efficient and they could require to be redefined by an external entity (e.g. the application itself or another component that is aware of the context change). For instance, when the delay in the communication becomes too high and the network is not reliable, retransmission performed by an error control mechanism could produce an accumulated delay that is not acceptable for the application.

In such a scenario for example, the retransmission policy could be configured to accept some losses when the delay is too important. But if network conditions improve in the future, the fully reliable policy could be reactivated. Likewise, if the conditions become even worse, the retransmission mechanism itself could be replace by another error control mechanism more efficient in this kind of network scenarios (e.g. Forward Error Control mechanism).

In order to be able to allow external behavioral or structural adaptation, the FPTP compositional architecture has been enhanced, in order to integrate an adaptive port and the

corresponding interface for receiving adaptation requests. Figure 1 illustrates the adaptive communication pattern proposed by ETP.

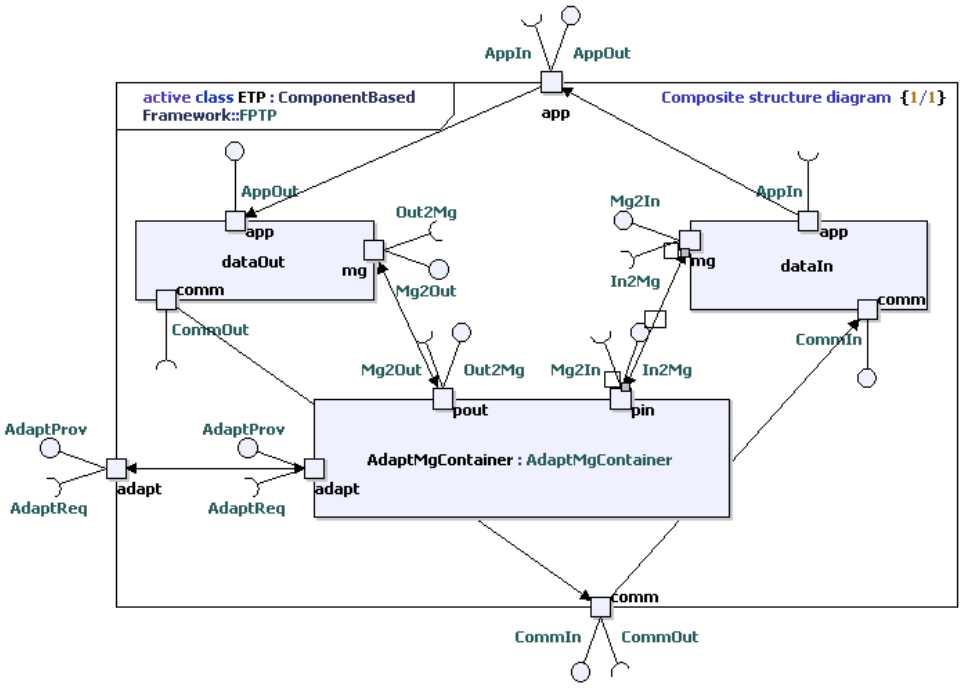

Figure 1. ETP adaptive architecture

This adaptive pattern provides a new port (named "adapt" port in Figure 1) allowing receiving adaptation requests for an external component. For instance, an application or any other external entity, could use this interface to adapt the error control strategies for lossy networks when the observed delay is too high.

Likewise, this port integrates a symmetric interface aimed at sending adaptation requests to external entities. For instance, when the management plane of an ETP transport protocol instance is not able to satisfy the application requirements, it could send an adaptation request to an external entity that has been assigned or connected to during the deployment phase. This external entity could be the application itself which could perform an application level adaptation strategy (e.g. reducing the quality of a multimedia video stream in order to reduce the required bandwidth). Likewise, the external entity could also be another transport entity which could collaborate with the ETP instance (e.g. by reducing the sending rate in order to leave some additional bandwidth to the other transport connection).

More information about the ETP architecture and the adaptive transport composite pattern can be found at [Van Wambeke 2007b], [Van Wambeke 2008b], [Exposito 2008a], [Exposito 2009a].

The adaptive architecture proposed by ETP also allows the design and implementation of behavioral and structural adaptation strategies.

2.3. Behavioral adaptation

Behavioral adaptation consists in dynamically tuning transport mechanisms in order to make them more efficient face to changing favorable or adverse network conditions. Requests for behavioral adaptation are received by transport mechanisms from an external management plane entity able to measure the current offered service and compare it with the application expectations. Based on this result and on the current environment conditions, the management entity may send adaptation requests to the adaptive transport mechanisms. Behavioral adaptation strategies available at the transport mechanism level would be too limited if information about the characterization of application data is not available.

In order to answer to these needs of application data characterization, we have designed and implemented a QoS application-level interpreter component aimed at being integrated within the ETP architecture. This QoS-interpreter is intended to make Application Data Units (ADU) properties of multimedia streams accessible and understandable by mechanisms operating at the transport layer. The QoS-interpreter offers a standard API (Application Programming Interface) to retrieve the QoS properties of ADUs for standard and proprietary multimedia streams. This section presents the design of the QoS-interpreter and explores the behavioral adaptive transport mechanisms that can be designed following this approach.

2.3.1. Design principle

In contrast with traditional distributed applications presenting basic requirements in terms of reliability and order, multimedia applications such as VoIP, VoD or IPTV, present more complex requirements and constraints which can only be satisfied by specialized transport services (e.g. a service composite resulting from partially-reliable and time-controlled functions). However, intrinsic requirements and constraints of the data units produced by applications are usually ignored by transport mechanisms. However, the QoS observed by the final user may be strongly related to the way individual application data units are transmitted and processed.

The difficulty in taking into account this important information is mainly due to large diversity of standard or proprietary multimedia codecs. However, if these properties were easily interpreted, they could be used within the transport mechanisms' algorithms, in order to optimize the rendered service.

Most of these applications follow the Application Level Framing approach (ALF) [Clark 1990]. ALF principle claims for breaking media data (i.e. audio or video content) into suitable aggregatescalled Application Data Units (ADU) representing the minimal processing unit. For traditional ALF-based multimedia codecs as well as for the last advanced audio and video streaming standard formats (i.e. MPEG4, H.264, etc), the QoS properties could be deduced using the information conveyed within the ADU headers.

Several related research works aimed at demonstrating the interest of using application layer information for adapting communication services have been carried out [Bolot 1996],

[Frossard 2001], [Vass 2001], [Ahmed 2005], [Johnsson 2005],[Van der Schaar 2005], [Ksentini 2006].

For instance, in [Ahmed 2005] a cross-layer video streaming system for MPEG4 is proposed. In this system, a cognitive layer provides adaptation capabilities based on the nature of the sub-streams in MPEG4 content. In [Ksentini 2006] a similar approach has been proposed, this time for the classification of H.264 streams in the context of QoS-oriented wireless networks. Likewise, works for scheduling algorithms optimization based on QoS properties such as the packet delay deadline has also been carried out [Johnsson 2005]. In [Frossard 2001], [Vass 2001] a diversity of mechanisms are proposed to take into account the hierarchical nature of video encoded streams to improve the QoS. Finally, in [Van der Schaar 2005] different cross-layer strategies adapted to wireless multimedia systems are presented.

Most of these approaches are restricted to using particular properties of specific media streams for improving isolated QoS mechanisms. These efforts fail in offering global solutions and are mainly focused to use particular fields of well-identified multimedia streams in order to implement ad-hoc solutions at a specific layer of the communication system.

Our work has been oriented to provide a cross-layering approach by allowing any mechanism at the transport layer to access and understand application layer information conveyed within the ADU headers. This information is intended to be accessed via an unique API provided by a generic QoS-interpreter. This interpreter is intended to be used by adaptive mechanisms able to take advantage of fine-grained ADU-level properties. This interpreter has been designed to be easily extended in order to integrate future codecs following the ALF approach.

2.3.2. Interpreter API

Most of the streams that follow the ALF approach are packetized in a way that QoS properties are transparently available at fixed locations within the ADU, generally assembled to form a header. In [Exposito 2008] and [Exposito 2009], the application-layer semantic model guiding the design and implementation of this interpreter for a large diversity of RTP media streams has been presented. Likewise, the rules guiding the recognition of generic QoS properties based on the standard RTP traffic profile specifications have been presented. These generic QoS properties can be categorized in three different groups: identification, prioritization and dependency:

- Identification properties are generally included as a way of identifying individual ADUs or groups of ADUs belonging to the same stream, as well as to recognize the type of stream and/or the nature of the multimedia session. The QoS-interpreter proposes a generic representation of identification characteristics by the following properties:
  - Unique ID: uniquely identifying every ADU within the same multimedia stream
  - ADU Type: allowing to identify the various classes of ADU (i.e. sub-streams) within the multimedia stream (e.g. I, P and B frames for MPEG4 video streams)
  - Stream Type: identifying the nature of the multimedia stream (e.g. audio, video, text, pictures, etc.)

    - Session Type: classification of the session based on its requirements (e.g. conversational or interactive, messaging, streaming, gaming, etc.)
- Priority related properties are most commonly defined for hierarchical media codecs (e.g. MPEG2, H.263, H.264, etc.). Moreover, when a stream results from the multiplexing of various data sources, the resulting packets might be assigned different priorities in order to differentiate the importance of each of the multiplexed streams. Furthermore, the maximum ADUs' lifetime can also allow deducing relative priorities. The QoS-interpreter enables standard access to this category of characteristics by defining the following properties:
    - ADU Priority: giving the relative priority of a type of ADU (e.g. I pictures are "more important" than P pictures in H.264 or MPEG video streams)
    - Presentation Time: providing an easy way to estimate the end-to-end tolerated delay for any given ADU (e.g. 150 milliseconds for interactive applications)
- Dependency related properties are aimed at expressing dependency relationship that might exist between groups of ADUs in the same stream or between streams belonging to the same multimedia session. In order to represent these dependency characteristics, the QoS-interpreter defines the following properties:
    - Intra-Dependency: expressing the dependencies between a set of ADUs representing a segmented application object (e.g. dependency between various segments of an I picture).
    - Inter-Dependency: aimed at expressing the dependency relationships between different classes of ADU (e.g. P pictures depend on I pictures)
    - Synchronization dependencies: intended to represent the dependencies between synchronized streams of a same applicative session (e.g. lips synchronization between audio and video stream for a videophony session)

This generic set of QoS properties proposed by the QoS-interpreter are aimed at providing an uniform interface for QoS mechanisms based on adaptive algorithms able to implement behavioral adaptation.

2.3.3. QoS adaptive mechanisms

The next paragraphs present a list of potential transport mechanisms and the ADU properties that could be used to perform QoS behavioral adaptation:

- Flow shaping: adaptation of the traffic profile based on the flow requirements and underlying resources. This mechanism could use the QoS-interpreter to limit the accumulated delay in order to respect the ADU presentation time constraints.
- Flow policing: definition of actions to be taken when the flow specification is violated. Using the QoS-interpreter, these actions or policies could be extended in order to respect the dependency and priority related properties (e.g. presentation time, priorities and intra/inter and synchronization dependencies).
- Flow synchronization: control of order and time requirements for the delivery of multiple streams. The QoS-interpreter could be used to define the synchronization policies in order to take into account the ADU-level presentation time as well as inter, intra and synchronization dependencies between related streams.

- Error control: including detection, reporting and recovery of errors by retransmission or redundancy. The retransmission process could be optimized using the unique ID, presentation time, types, priorities as well as inter, intra and synchronization dependencies. Using these properties could help avoid retransmission of obsolete or less important ADU while respecting their dependencies constraints.

In [Exposito 2008], [Exposito 2009], [Exposito 2009c], [Gineste 2009] and [Gineste 2009a], several studies have been carried out in order to evaluate the advantages of designing and developing generic and adaptive transport mechanisms based on the semantic offered by the QoS-interpreter. These studies have shown how adaptive mechanisms (i.e. error and congestion control) have been implemented by QoS interpreter based algorithms independently of the multimedia codec used at the application layer. Moreover, these experiments have shown how these mechanisms have been able to provide a more efficient behavior in varying network conditions.

In order to coordinate behavioral adaptation strategies and to extend them with the required structural adaptation strategies in dynamic network scenarios, we have also worked in designing a model-driven approach. This approach will be presented in the next paragraphs.

2.4. Structural adaptation

In highly dynamic network environments, behavioral adaptation strategies are limited to the adaptability spectrum provided by the algorithms implemented by the deployed transport mechanisms. Indeed, adaptive mechanisms could encounter extreme environment conditions for which their adaptation strategies are not able to provide the expected behavior. In these cases, structural adaptation strategies aimed at selecting and deploying more adapted transport mechanisms need to be applied.

Structural adaptation can be achieved by replacing one or several of the deployed transport mechanisms with more efficient implementations able to better answer to the observed environment conditions. The components replacement needs to be performed following an efficient workflow guiding a sequence of phases for structural adaptation. This workflow comprises an initial phase for the deployment of the new components followed by a transitional phase where both configurations are concurrently enabled in order to avoid service interruption and a final phase where the inefficient mechanisms are undeployed. This workflow needs to be smoothly executed in order to drive the adaptation process from behavioral to structural thus reducing or minimizing the impact in the underlaying system and without perturbing the final service users.

We have studied two different approaches to perform structural adaptation. The first approach is based on an analytic model able to drive the adaptation process [Van Wambeke 2008]. The second approach proposes the use of a learning-based model in order to guide the structural adaptation in scenarios where an analytic model cannot be built [Van Wambeke

2009]. Both approaches are intended to provide model-driven structural adaptation capabilities. Next paragraphs will introduce theses approaches.

2.4.1. Analytic model-driven adaptation

The analytic model-driven approach is based on the definition of two models representing the transport mechanisms composition and the adaptation decision process. These models allow enforcing runtime adaptation decisions based on the set of valid transport mechanisms compositions and the current state of the network.

- The composition model is aimed at defining the conditions of validity of the transport mechanisms compositions. These conditions are provided by the protocol designer and include the rules guiding all the valid combinations of mechanisms providing the required functions to implement the various services offered by ETP. This model needs to provide the required semantic to describe the list of the available mechanisms as well as their complementary or conflictive relationships with other mechanisms. Complementary relationships can exist between mechanisms able to be composed to provide a composite function (e.g. error detection and error recovery mechanisms); conflictive relationships mean that the mechanisms cannot be composed or that a specific composition order need to be respected in order to avoid conflicts in their operations (e.g. full and partial reliable mechanisms in a same composition).
- The decision model: this model allows the selection of one of the valid mechanism compositions in order to maximize the services offered to the final user for the current network environment. This model should be able to provide the required semantic to decide at which stage the behavioral adaptation of a mechanism has to be considered as inefficient and that a structural adaptation needs to be carried out. Actually, structural adaptation may be more expensive than behavioral adaptation in terms of system performance and resources. For this reason, an efficient decision model is required to trigger structural adaptation strategies.

The definition of an analytical model including both composition and decision models and adapted to the ETP transport architecture has been presented in [Van Wambeke 2008]. This analytical model has been applied to the composite and adaptive transport mechanisms offered by ETP in order to support structural adaptation based on the observed environment. This approach has allowed to address the automated selection of internal composition of transport mechanisms. In order to design and implement this model, a fundamental monitoring component able to model the observed environment in terms of QoS parameters has been incorporated in the protocol architecture.

This analytical model-driven approach has been experimentally evaluated for multimedia applications in the context of a wireless networks. Results of this experiment have been compared with all possible candidate compositions declared valid by the composition model. This study has shown that the dynamic solution provided by the model was coherent with the expected structural adaptation strategies to be enforced based on the network conditions.

Further information including formal specification of composition and the decisional model as well as the experimental studies can be found at [Van Wambeke 2007], [Van Wambeke 2008] and [Van Wambeke 2008a].

Other studies aimed at studying and applying model-driven adaptation strategies at different levels of the communication system have also been carried out. For intance, in [Guennoun 2008], a framework for architecture-centric models supporting automated and adaptive deployment of communication services for QoS-enabled end-to-end group communication systems has been proposed. Likewise, in [Chassot 2006] and [Armando 2007], a model-based approach aimed at guiding adaptation strategies at transport, middleware and application layers has been presented.

2.4.2. Learning-based model-driven adaptation

Limitations to the analytic model approach can be found when the modeled system and its relationships with the environments are to complex to be determined and represented. In these cases other methods are required to perform structural adaptation.

We have studied methods based on intelligent control (e.g. learning-based control) in order to guide structural adaptation strategies for this kind of contexts [Van Wambeke 2009]. In this approach, the decision is based on an extension of the Markov Decision Process formalism (MDP) called eMDP. eMDP has been designed and implemented in order to dynamically adapt the transport mechanisms composition to the observed network environment. This eMDP is obtained through reinforcement learning technique.

A set of experiments has been carried out in order to evaluate the efficiency of this approach in order to perform the required adaptation aimed at satisfying application requirements under a changing environment emulated by varying network conditions. These experiments have illustrated the benefits of such approach for QoS constrained applications.

2.5. Conclusions and lessons learned

In this section the adaptive composite architecture as well as the behavioral and structural adaptation strategies promoted by ETP have been presented. This enhancement to the FPTP protocol has allowed to integrate in ETP the required runtime adaptation capabilities in order to face to dynamic network environments.

At this stage and according to the autonomic maturity level proposed in [Kephart 2003], the ETP protocol can be classified as being in the adaptive phase.We have initiated new studies aimed at promoting this component-based and adaptive transport architecture in order to design and implement a new version providing autonomic capabilities.

In the next section, a state of the art of the Autonomic Computing paradigm will be introduced in order to guide the design of the autonomic enhancement of ETP.

3. State of the art on autonomic computing

3.1. Introduction

The autonomic computing (AC) approach has been proposed by IBM in order to face the increasing complexity of manual management of information technologies by incorporating self-managing capabilities within software systems [Horn 2001], [Kephart 2003], [AC 2006].

The term autonomic has greek origin and means "self-governed". The AC approach is inspired by the human autonomic nervous system which is responsible for controlling the vital body functions without explicit conscious effort. In the case of software systems, the AC approach targets the implementation of self-managing functions by requiring a minimal intervention of users.

The AC paradigm proposes a specialized architecture based on a set of well defined components and interfaces as well as a precise specification of the individual and collective expected autonomic behavior of the different system components. AC systems are intended to autonomously operate according to policies specified by system users and/or administrators. Furthermore, autonomic knowledge sources need to be created and collected by AC systems.

Autonomic computing cannot be considered as a radically different new approach but rather as the integration of well developed research fields and elaborated theories and techniques from diverse areas such as control theory, adaptive systems, distributed and real-time systems, software agents, fault tolerant computing, machine learning, robotics, etc. [Muller 2006]

Since the IBM autonomic computing initiative in 2001, an important number of industrial organizations have been actively collaborating in designing and developing AC systems. Examples of these efforts are: The Dynamic Systems Initiative (DSI) of Microsoft, the Adaptive Enterprise of Hewlett Packard, the Grid Engine of Sun (SGE), the Dynamic Computing Initiative of Dell, etc.

More recently, the AC paradigm has also been applied to the area of network resources and technologies management. The autonomic networking (AN) approach is aimed at reducing the complexity involved network resources and services management face to the increasing diversity of heterogeneous networked devices, distributed applications and network technologies [Strassner 2006], [Dobson 2006], [Dobson 2010]. Likewise, the autonomic communication paradigm has been proposed based on the AC vision and adapted to distributed communication services [Smirnov 2004].

3.2. Self-managing functions

Self-managing funcions of an autonomic system (AS) are aimed at implementing adaptation actions resulting from changes or events observed in the environment and intended to keep

offering an adequate service [Sterritt 2005], [Muller 2006], [Sterritt 2007]. Adaptation actions are implemented by adaptive algorithms operating within a closed-loop control system.

The AC framework classifies self-managing functions in four categories:

- Self-configuring: an AS should be able to configure and reconfigure itself in response to a dynamic and unpredictable environment in order to offer an adequate service that satisfies expected functional or non-functional requirements. Several kinds of self-configuration capabilities can be implemented based on operational rules or expected goals. Reconfiguration could be achieved by the deployment of new components or the undeployment of active components.
- Self-optimizing: An operational self-configured AS facing changes in its environment needs to tune itself and perform adaptive actions in order to provide an efficient service able to satisfy the expected non-funtional requirements.
- Self-healing: An AS should be able to detect and prevent service interruption and perform actions to recover itself from a malfunctional state.
- Self-protecting: An AS should be able to implement protection actions in order to detect and avoid attacks from its environment.

3.3. Architecture

The architecture promoted by the autonomic computing framework is based on elementary entities called autonomic elements (AE) [Kephart 2003], [White 2004], [Sterritt 2005], [Muller 2006]. An autonomic element results from the composition of a basic non-autonomous component called managed element (ME) and a controller able to manage that element and called autonomic manager (AM).

3.3.1. Autonomic Elements

Autonomic elements are defined by the interconnection of managed elements and autonomic managers within a closed loop control system. Indeed, autonomic systems operate based on a control loop system where an AM receives inputs from its ME and answers with outputs indicating the actions that the ME has to perform in order to maintain a desired property of the autonomic system.

The interface between the autonomic manager and the managed element is called the touchpoint interface. This management interface is composed of the sensor and the effector interfaces:

- The sensor interface allows the AM to retrieve information about its managed ME. Sensor interfaces offer two operational modes: request-response or subscription for alerts or notifications.
- The effector interface allows the AM to execute actions on the ME. Two operational modes are offered by this interface: synchronous execution or callback based asynchronous execution.

Figure 2 illustrates the AE internal composite architecture.

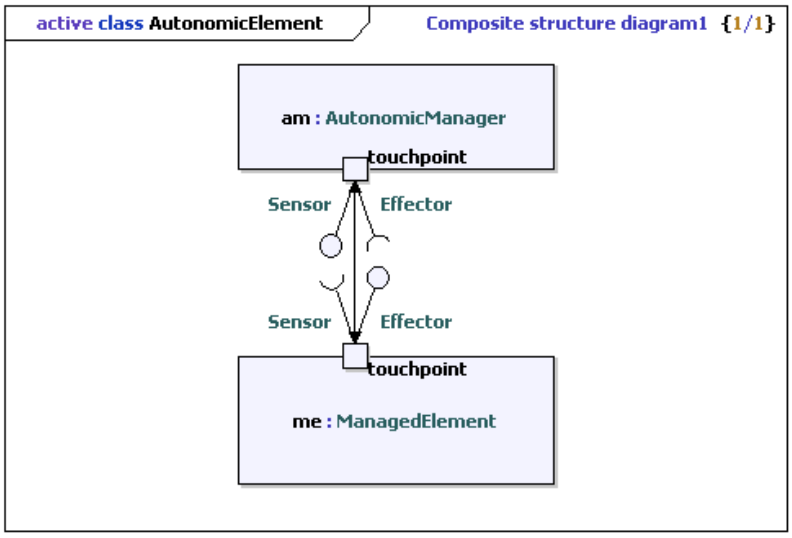

Figure 2. Autonomic element composite architecture

In order to implement the adequate management functions, the AM implements a workflow based on monitoring, analyzing, planning and executing (MAPE) activities sharing a common knowledge base.

- Monitoring: retrieves state information of the AE via the sensor interface. Relevant information is filtered and stored in the knowledge base.
- Analyzing: compares the observed data from the expected values in order to detect an undesirable state.
- Planning: selects or elaborates strategies aimed at preventing or correcting an undesirable state or intended to achieve the targeted goals.
- Executing: executes the tuning actions on the AE via the effector interface and traces this information in the knowledge base for future analysis and planning.

Figure 3 presents a diagram illustrating the AC architecture. In this diagram, an autonomic element is composed of an autonomic manager and a managed element, implementing and requiring their respective sensor and effector interfaces. The autonomic manager interface provides the operations to be implemented by the MAPE functions based on a knowledge base

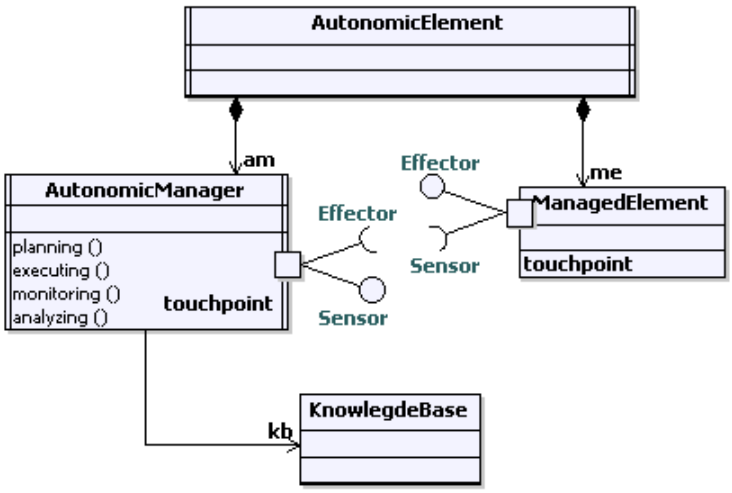

Figure 3. Autonomic computing architecture

The diagram presented in Figure 4 illustrates several compositions AM and ME components.

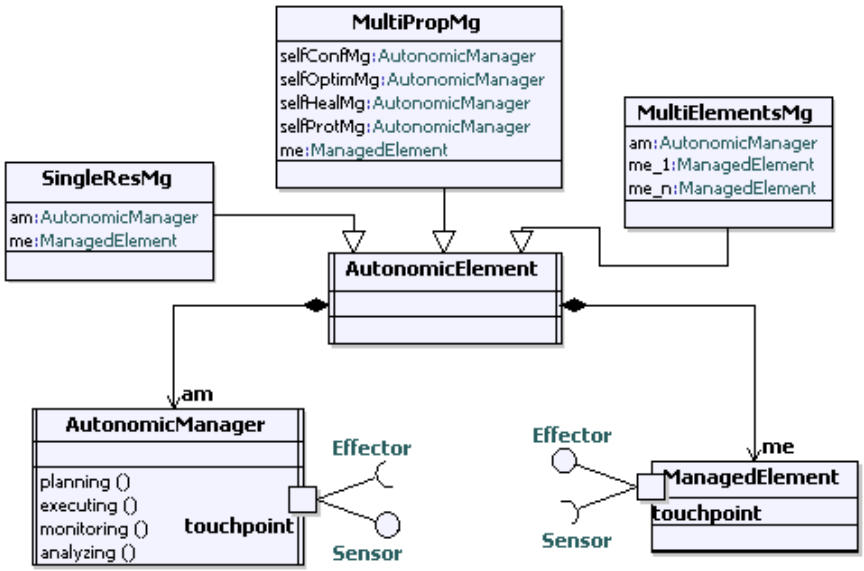

Figure 4. Autonomic element compositions

Several composition can be defined by one or several AM and one or several ME:

- single resource management: one AM managing one ME
- multi-properties management of a single resource: several AM managing the various properties of one ME (i.e. self configuring, self-optimizing, etc).
- multi-elements management: one AM managing a homogeneous or heterogeneous set of ME.

3.3.2. Autonomic orchestrators

Similarly, at higher levels of an autonomic system, hierarchical compositions of autonomic elements and autonomic managers can be defined. In these compositions, each managed AE offers the sensor interface and implements the effector interface. A higher level AM manages the lower level AE by implementing the MAPE functional phases and communicating with the AE via the touchpoint interface.These high level autonomic managers are called orchestrators (see Figure 5).

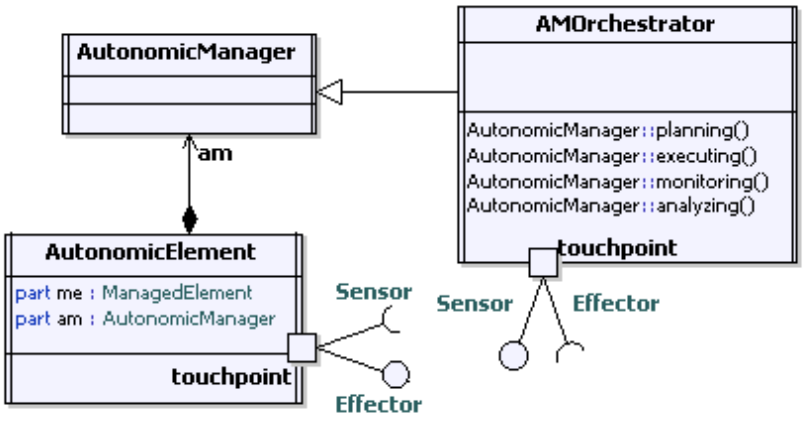

Figure 5. Autonomic manager orchestrator definition

At lower or higher levels of an autonomic system, the overall self-management functions (i.e. self-configuring, self-optimizing, self-healing and self-protecting) as well as the MAPE activities performed by basic autonomic managers orchestrators are fundamentally based on its knowledge base and the policies guiding its autonomic behavior.

3.4. Policies

An AS is able to perform self-managing functions based on the policies guiding its autonomic behavior.

In [Westerinen 2001] a policy is defined as a definite goal, course or method of action to guide and determine present and future decisions. Moreover, policies can also be defined as a set of rules to administer, manage, and control access to network resources [Moore 2001], [Moore 2003]

In [Kephart 2004], a framework aimed at defining and implementing AC policies have been proposed. This framework is based on the concept of rational agents defined in the artificial intelligence domain. Rational agents are defined as entities able to perceive and to act in response to change of its environment. In the AC framework, autonomic elements perform as rational agents and select the adequate actions based on its knowledge base in order to maximize the element's objective.

In this framework, three categories of AC policies have been considered:

- Action policies: based on an "if-then" rules structure, action policies guide an AM in the process of selecting the adequate action that transitions the system from its current state.
- Goal policies: instead of providing explicit actions to be executed for specific states, goal policies express the expected state of the system. The goals can also be indirectly described by specifying a set of expected properties that characterize a set of desired states.
- Utility-function policies: this category generalizes goal policies by providing an objective function able to express the utility values for each possible state of the system.

More information about autonomic policies definition can be found at [White 2004], [Kephart 2005], [Pena 2006], [Huebscher 2008], [Cheng 2008]

3.5. Knowledge base

An AS operates based on knowledge bases. An AM uses its own knowledge base or can share it to or import it from others AM. Moreover, the AS itself can create new knowledge based on the context observation and the results obtained from its actions.

A knowledge base is composed of structured data providing the required syntaxis and semantic of symptoms, policies, goals, measurements, plans or change requests. This knowledge is classified in several kind of semantic information:

- Topology knowledge: contains information about the components, their configuration and instantiation. This information is used when configuring or reconfiguring the system.
- Policy knowledge: integrates decision-making rules used to trigger actions guiding self-managing functions of the system.
- Problem determination knowledge: includes measurements, symptoms and decision trees and it is used to manage the system.

More information about knowledge bases definition, sharing and creation can be found at [Clark 2003], [Huang 2005], [Kephart 2005].

3.6. Summary

The AC paradigm presented in this section, proposes a well suited framework able to enhance adaptive mechanisms by providing self-managed transport elements. Based on the lessons learned from the design and implementation of the adaptive ETP protocol and the architectural and functional decomposition of self-managen functionalities promoted by the autonomic computing paradigm, the next section proposes the main guidelines leading the

design of an Autonomic Transport Protocol (ATP). Likewise, several recommendations for designing an Autonomic Transport Layer (ATL) able to select, compose, deploy and orchestrate transport protocols based on the autonomic computing paradigm and integrating the semantic-driven, service-component and service-oriented approaches presented in the previous chapter will also be presented.

4. Design guidelines of an autonomic computing architecture for the next generation transport layer

This section is aimed at proposing several guidelines for designing an autonomic transport layer of next generation. These guidelines are based on the lessons learned from the design and development of the adaptive Enhanced Transport Protocol and the framework proposed by the autonomic computing paradigm.

4.1. Self-managing functionalities

- Self-configuring: An autonomic transport protocol should be able to implement self-configuring functionalities guiding the selection and deployment of the adequate transport mechanisms. A predefined configuration aimed at providing a minimum set of basic transport mechanisms such as multiplexing/demultiplexing, segmentation/reassembly, data integrity and connection management should be provided. This basic configuration should be enhanced in order to provide specialized network-aware data-control and management functions taking into account the underlying communication services (i.e. Best-Effort or guaranteed services, wireless or wired networks, etc). These network-aware transport functions could include congestion control or rate control functions. Likewise, application-aware functions such as error control mechanisms aimed at satisfying application requirements should be incorporated. The adequate managers aimed at guiding runtime adaptation strategy should also be integrated. An autonomic manager able to measure and analyze changes in the environment and in consequence able to plan and execute behavioral or structural adaptation needs to be incorporated in the final transport composite service.
- Self-optimizing: An autonomic transport protocol should be guided by an autonomic manager able to perform self-optimizing functions. This manager should be able to request adaptation actions in order to perform behavioral adaptation. Adaptive transport mechanisms able to execute adaptation actions aimed at improving the service offered to the applications while respecting network constraints should be integrated as managed elements of an autonomic transport protocol.
- Self-healing and self-protecting for unexpected events: These functionalities should also be incorporated in an autonomic transport protocol. At this stage, our studies have been restricted to self-configuring and self-optimizing features, but further studies should be carried out to enhance the current design in order to integrate these additional functionalities.

4.2. Architecture

4.2.1. Autonomic elements

An autonomic transport protocol following the ATP architecture should be designed based on the following elements:

- Autonomic elements: An Autonomic Transport Protocol (ATP) resulting from the composition of one transport managed element implemented by a composition of ETP mechanisms and an autonomic manager able to drive the behavioral and architectural adaptation strategies should be designed to conform the autonomic element specification.
- Managed elements: ETP adaptive transport functions result from the composition of data-control and management planes mechanisms able to implement adaptation requests received from the adaptive port. The adaptive interface proposed by ETP should be generalized by the autonomic touchpoint interface in order to implement sensor and effector interfaces. ETP mechanisms commonly require to perform internal monitoring of the service provided and the network conditions in order to implement their specific algorithms. These monitoring measurements should be available to autonomic managers via the sensor interface. Likewise, ETP mechanisms operating at the management plane should be able to process action requests received from autonomic managers via the effector interface. In this way, ETP adaptive mechanisms will conform the managed element specification.
- Autonomic managers: Autonomic transport managers able to monitor and analyze information received from ETP managed elements and to plan and execute adaptation actions should be integrated in an autonomic transport architecture in order to build autonomic transport elements.

Figure 6 illustrates the ATP composite architecture resulting from the composition of ETP managed elements and transport autonomic managers.

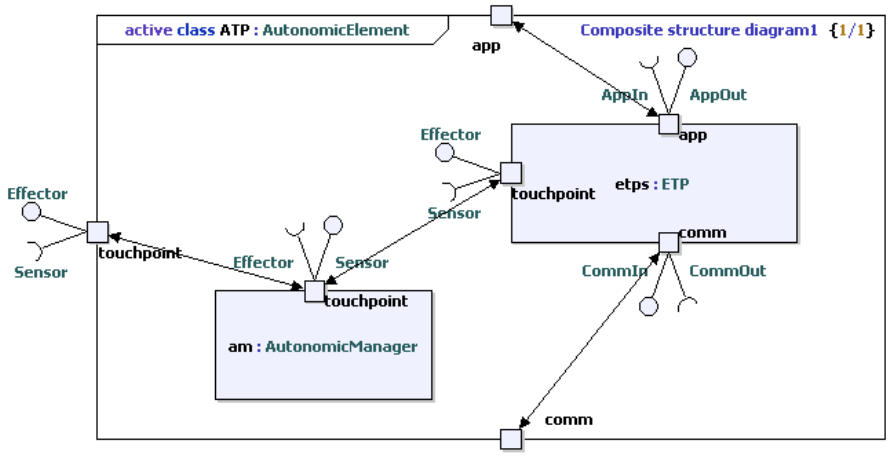

Figure 6. Autonomic transport protocol architecture

Figure 7 illustrates the evolutive design process that has led us to propose the design of the autonomic transport protocol ATP, based on the composition of autonomic managers together with the adaptive ETP mechanisms compositions (already being built following the component-based architecture of FPTP).

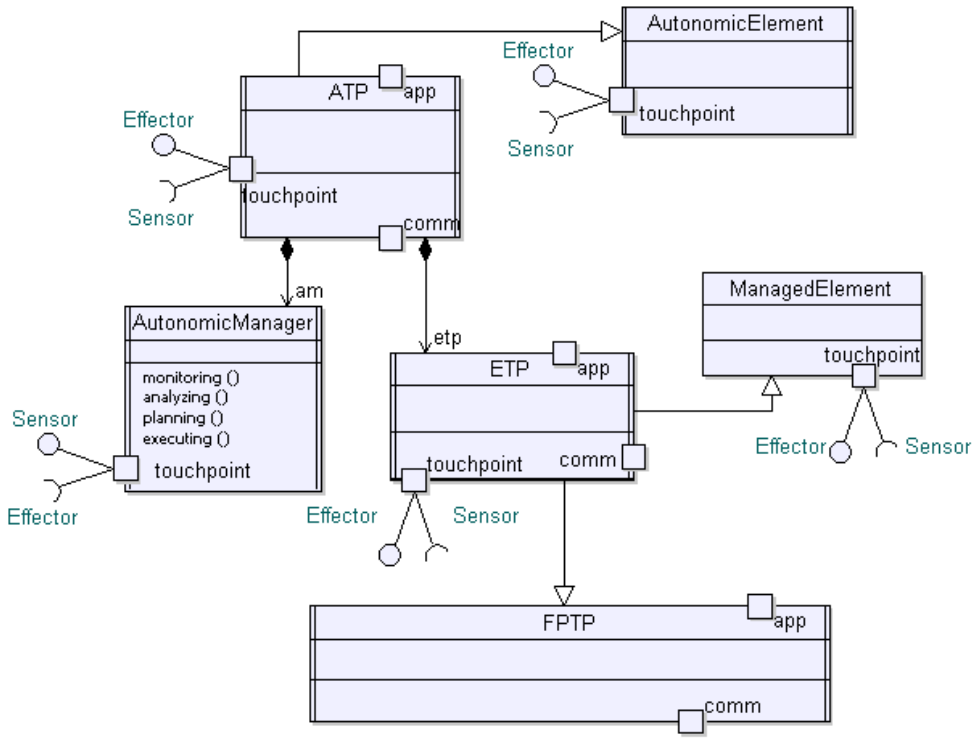

Figure 7. Autonomic transport protocol definition

Figure 8 presents an example aimed at illustrating the relationships between FPTP, ETP and ATP protocols.

In this example, a composed partially reliable service based on the composite FPTP architecture has been extended in order to design an adaptive partially reliable service based on the ETP adaptive architecture. This adaptive service has been composed with a time control autonomic manager in order to implement an autonomic partial reliable service based on the autonomic element architecture provided by ATP.

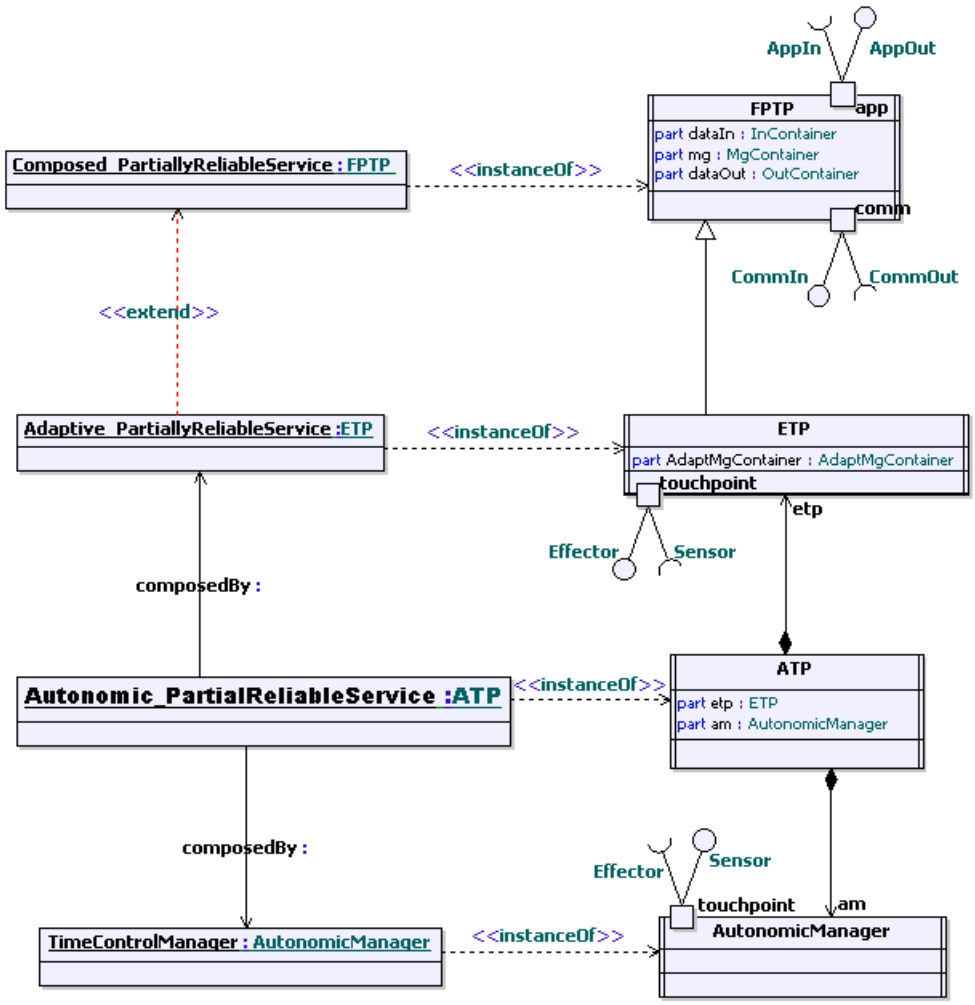

Figure 8. FPTP, ETP and ATP protocols relationships

In this example, the adaptive partially reliable service is aimed at being managed by the time control manager based on policies expressing application preferences and requirements. For instance, based on accumulated delay measures retrieved from the sensor interface, the time control manager could send adaptation requests to the adaptive mechanisms via the effector interface, in order to increase the tolerated partial reliability when the delay is too high. Likewise, this manager could decide to decrease the partial reliability tolerance when the network conditions improve.

4.2.2. Autonomic orchestrators

Autonomic element proposed in the previous paragraphs could be coordinated following the autonomic orchestration approach proposed by the AC paradigm. These autonomic orchestrators will provide the basic coordination functionalities required by an Autonomic Transport Layer (see Figure 9).

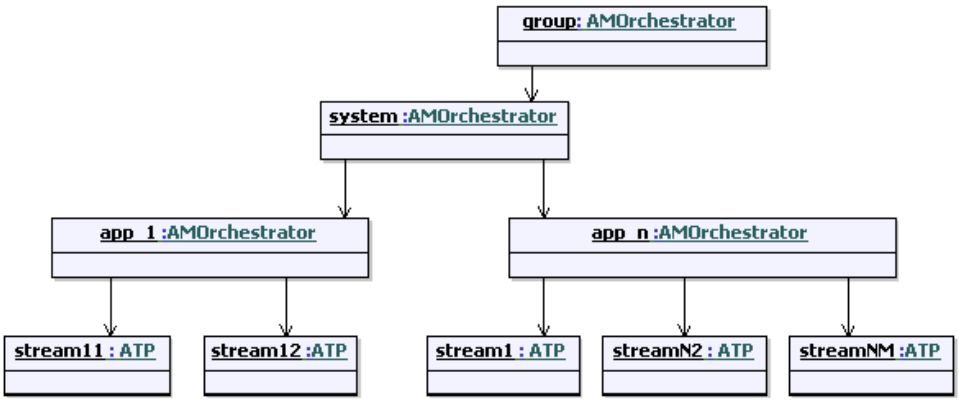

Figure 9. Autonomic manager orchestration

- Stream-level: managers operating at stream level (e.g. in Figure 9 see the lowest level of autonomic managers) correspond to the fundamental autonomic elements represented by ATP instances. Behavioral adaptation strategies based on the QoS ADU properties provided by the QoS-interpreter could be implemented by adaptive mechanisms composing the deployed ATP service. Likewise, model-driven structural adaptation strategies could be performed in order to face to important changes of the network conditions.

- Application-level: managers orchestrating the several streams of the same application could be deployed at this level. This autonomic manager takes into account the application requirements or preferences in order to manage the service offered to each stream (e.g. audio/video synchronizer or bandwidth optimizer orchestrators).

- System-level: managers orchestrating the various application-level manager within the same end-system. This orchestrator is based on user requirements and preferences in order to manage the service offered to each application.

- Group-level: managers aimed at enforcing adaptation policies between the various end-systems composing a collaborative group and following group requirements and preferences (e.g. in Figure 9 see the highest level of autonomic managers). The collaboration can be defined for instance for sharing network resources.

This category of autonomic manager orchestrators illustrates the large scope of transport services management and coordination that can be achieved by following this approach. Stream-level management is the responsibility of autonomic transport protocols (autonomic elements). Application, system and group levels are the responsibility of autonomic orchestrators based on applications, users or groups requirements and preferences respectively.

These managers are aimed at implementing the required behavioral or structural adaptation strategies in order to satisfy the overall requirements of system actors (i.e. application, users and groups of users). At the lowest level (stream-level), behavioral of structural adaptation can be performed on transport mechanisms. At higher levels (application, system or group levels), structural adaptation can be performed on autonomic elements represented by instances of the autonomic transport protocol.

The example shown in Figure 10 is aimed at illustrating autonomic manager orchestration at application and system levels.

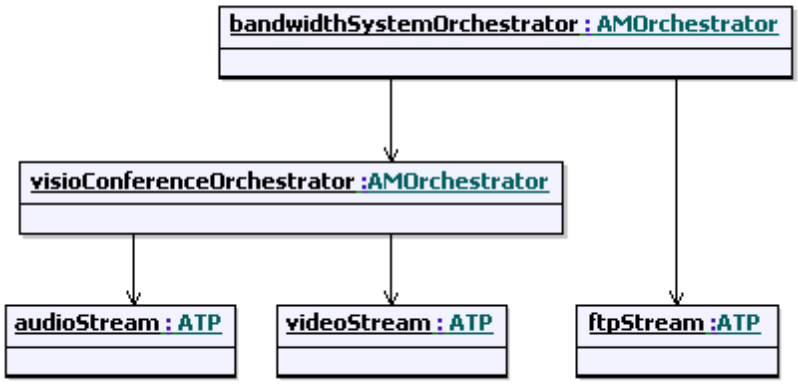

Figure 10. Application and system levels orchestrators example

In this example, several applications operate concurrently sharing the available network resources of an end-system. A time-constrained visio-conferencing application share the available resources with a downloading application.

a) For the visio-conferencing application, an application-level orchestrator is responsible for audio and video streams synchronization. For each stream, specific transport mechanisms have been deployed within the ATP instances. The visio-conference orchestrator could require to send adaptation requests if the measured delay for each stream shows that multimedia synchronization constraints are not respected (e.g. audio/video lips synchronization). Adaptation requests could be translated in behavioral adaptation implemented by adaptive mechanisms based on the specific audio or video stream properties provided by the QoS-interpreter. Structural adaptation aimed at deploying more adapted mechanisms could also be performed if necessary.

b) At the system-level, the bandwidth orchestrator could help the visio-conference orchestrator to achieve its goals. Indeed, based on the user preferences, the bandwidth orchestrator could identify the visio-conference orchestrator as having a higher priority for bandwidth consumption with respect to the downloading application. In consequence, if the bandwidth system orchestrator receives an alert from the visio-conference orchestrator, it could send an adaptation request to the ATP instance providing the transport services of the ftp application, in order to reduce bandwidth usage. At the

stream-level, the autonomic manager controlling the ATP service provided to the ftp application will send request alerts to the deployed adaptive transport mechanisms. For instance, an adaptive rate control mechanism could implement a behavioral adaptation by reducing the allowed rate.

c) Such a hierarchical coordination of orchestrators could also be deployed at a group-level. For instance, a group of users within a peer-to-peer network could decide to share their resources in a collaborative way. Group-level orchestrators will be responsible to coordinate the orchestrators operating on each system of the group. System-level orchestrators will be responsible to coordinate their application orchestrators while helping the group orchestrator to implement collaborative self-management functions.

Finally, a common knowledge base and policy framework aimed at guiding self-managing and adaptive functionalities of transport protocols as well as the different level of services orchestration functionalities of an autonomic transport layer should be designed.

4.3. Policy framework

An autonomic transport protocol should integrate an efficient policy framework aimed at guiding the required behavioral and structural adaptation strategies. We have carried out several studies aimed at developing a goal-based policy framework based on analytic and learning-based models have been carried out [Van Wambeke 2008]. Likewise, other studies intended to develop utility-oriented policy framework has been started [Exposito 2010], [Gomez 2010]. Currently, we have initiated new studies aimed at defining this common policy framework.

4.4. Knowledge base

A knowledge base for the next generation transport layer should incorporate a QoS oriented semantic model aimed at integrating a common vision of application requirements and transport services in order to facilitate service selection and deployment. Likewise, the characterization of composite transport functions and the available transport mechanisms, as well as their adaptive capabilities should be incorporated in this base.

An extension of the QoS transport ontology model developed in chapters 1 and 2 could provide the required semantic to design such knowledge base. We are currently working in designing and implementing this semantic model.

5. Conclusions

In this chapter the autonomic computing paradigm including the promoted architecture and the self-managing functionalities has been introduced. This paradigm is well suited to self-

manage adaptive components based on the monitored environment conditions and guided by the service user policies.

Based on this paradigm, this study has presented our work on designing and developing the adaptive Enhanced Transport Protocol (ETP) resulting from the specialization of the FPTP component-based transport protocol. ETP has been designed to provide an adaptive-oriented architecture facilitating dynamic behavioral or structural adaptation based on the perceived network conditions. A specialized mechanism aimed at interpreting application requirements, preferences and constraints in order to guide behavioral adaptation strategies has also been incorporated in ETP. Likewise, our work in designing and implementing a model-driven approach to guide structural adaptation based on the reconfiguration of transport protocols has also been presented.

Several guidelines aimed at enhancing this component-based and adaptive transport architecture in order to design and implement an Autonomic Transport Protocol (ATP) have been presented. ATP will provide an autonomic computing architecture able to integrate knowledge base, policy orientation as well as self-managing functionalities. Behavioral and structural adaptation strategies promoted by ETP will be redesigned following the autonomic computing paradigm in order to facilitate self-managing functionalities. Likewise, studies aimed at designing an Autonomic Transport Layer (ATL) able to select, compose, deploy and orchestrate transport protocols following the autonomic computing paradigm and integrating the semantic-driven, service-component and service-oriented approaches will be carried out.

6. Annexes

- The ETP and ATP UML specifications can be found at http://homepages.laas.fr/eexposit/hdr/atp/uml/

7. References

[AC 2006] An Architectural Blueprint for Autonomic Computing, IBM white paper. (Online). Available: http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf, June, 2006

[Ahmed 2005] Y. Iraqi, R. Boutaba, A. Mehaoua, T. Ahmed, Adaptive packet video streaming over IP networks: a cross-layer approach, elected Areas in Communications, IEEE Journal on, 23, 2, February, 2005

[Armando 2007] Francois Armando, Nicolas Van Wambeke, Christophe Chassot, Ernesto Exposito, Khalil Drira, A framework for the dynamic configuration of adaptive transport protocols, International Conference on Wireless Information Networks and Systems (WINSYS 2007), Barcelone (Espagne), 28-31 july, 2007

[Bolot 1996] J-C. Bolot, T. Turletti, Adaptive error control for packet video in the Internet, IEEE Int. Conf. on Image Processing (ICIP'96), Lausanne, Switzerland, September, 1996

[Chassot 2006] Christophe Chassot, Karim Guennoun, Khalil Drira, Francois Armando, Ernesto Exposito, André Lozes, Towards autonomous management of QoS through model-

driven adaptability in communication-centric systems, International Transactions on Systems Science and Applications (ITSSA), 2, 3, 2006

[Cheng 2008] Y. Cheng, A. Leon-Garcia, And I. Foste, Towards an autonomic service management framework: A holistic vision of SOA, AON, and autonomic computing, IEEE Communications Magazine, 46, 5, May, 2008

[Clark 1990] Clark, D. D. and Tennenhouse, D. L., Architectural considerations for a new generation of protocols, ACM SIGCOMM Computer Communication Review, 20, 4, 1990

[Clark 2003] J. T. Wroclawski, J. C. Ramming, C. Partridge, D. Clark, A knowledge plane for the internet, SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, Karlsruhe (Germany), 25-29 August, 2003

[Dobson 2006] S. Dobson, S. Denazis, A. Fernandez, D. Gaiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, F. Zambonelli, A survey of autonomic communications, ACM Transactions on Autonomous and Adaptive Systems (TAAS), 1, 2, 2006

[Dobson 2010] M. Hinchey, P. Nixon, R. Sterritt, S. Dobson, Fulfilling the Vision of Autonomic Computing, IEEE Computer, 43, 1, 2010

[Exposito 2008] Ernesto Exposito, Nicolas Van Wambeke, Christophe Chassot, Khalil Drira, Introducing a cross-layer interpreter for multimedia streams, Computer Networks, 52, 6, 2008

[Exposito 2008a] Ernesto Exposito, Nicolas Van Wambeke, Design principles for an autonomic transport protocol framework, Latin American Autonomic Computing Symposium (LAACS 2008), Gramado (Brasil), 8-9 September, 2008

[Exposito 2009] Ernesto Exposito, Mathieu Gineste, Laurent Dairaine, Christophe Chassot, Building self-optimized communication systems based on applicative cross-layer information, Computer Standards & Interfaces, 31, 2, 2009

[Exposito 2009a] Ernesto Exposito, Jorge Gomez Montalvo, Myriam Lamolle, Semantic and Architectural Framework for Autonomic Transport Services, 1st International Conferences on Adaptive and Self-adaptive Systems and Applications, Athènes (Grèce), 15-20 Novembre, 2009

[Exposito 2009c] Ernesto Exposito, Mathieu Gineste, Myriam Lamolle, Jorge Gomez Montalvo, Semantic network adaptation based on QoS pattern recognition for multimedia streams, International Conference on Signal Processing, Image Processing and Pattern Recognition (SIP 2009), Jeju Island (Corée), 10-12 Décembre, 2009

[Exposito 2010] Ernesto Exposito, Jorge Gomez Montalvo, An ontology based framework for autonomous QoS management in home networks, The 6th International Conference on Networking and Services (ICNS 2010), Cancun (Mexique), 7-13 Mars, 2010

[Frossard 2001] O. Verscheure, P. Frossard, Joint source/FEC rate selection for quality-optimal MPEG-2 video delivery, Image Processing IEEE Transactions on, 10, 12, December, 2001

[Gineste 2009] Mathieu Gineste, Nicolas Van Wambeke, Ernesto Exposito, Christophe Chassot, Laurent Dairaine, A cross-layer approach to enhance QoS for multimedia aplications over satellite, Wireless Personal Communications, 50, 3, 2009

[Gineste 2009a] Mathieu Gineste, Nicolas Van Wambeke, Ernesto Exposito, Enhancing TFRC for video streaming by agnostically using applicative cross layer semantics and measure, 2nd International Workshop on Future Multimedia Networking (FMN 2009), Coimbra (Portugal), 22-23 June, 2009

[Gomez 2010] Jorge Gomez Montalvo, Ernesto Exposito, A semantic approach to user-based QoS provision for multimedia services in home networks, The Third International Conference

on Communication Theory, Reliability, and Quality of Service (CTRQ'2010), Athens/ Glyfada (Greece), June 13-19, 2010

[Guennoun 2008] Karim Guennoun, Khalil Drira, Nicolas Van Wambeke, Christophe Chassot, Francois Armando, Ernesto Exposito, A framework of models for QoS-oriented adaptive deployment of multi-layer communication services in group cooperative activities, Computer Communications, 31, 13, 2008

[Horn 2001] P.Horn, AutonomicComputing: IBM's Perspective on the State of Information Technology, IBM Research. (Online). www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf, 15 October, 2001

[Huang 2005] P. Steenkiste, A. Huang, Building Self-adapting Services Using Service-specific Knowledge, IEEE High Performance Distributed Computing (HPDC), Research Triangle Park, NC, USA, 24-27 July, 2005

[Huebscher 2008] J. McCann, M. Huebscher, A survey of autonomic computing -- degrees, models, and applications, ACM Comput. Surv., 40, 3, 2008

[Johnsson 2005] D. Cox, K. Johnsson, An Adaptive Cross-Layer Scheduler for Improved QoS Support of Multiclass Data Services on Wireless Systems, IEEE Journal on Selected Areas in Communications, 23, 2, February, 2005

[Kephart 2003] J. Kephart, D. Chess, The Vision of Autonomic Computing, IEEE Computer Magazine, 36, 1, 2003

[Kephart 2004] J. Kephart, W. Walsh, An Artificial Intelligence Perspective on Autonomic Computing Policies, 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004), Yorktown Heights, NY, USA, 7-9 June, 2004

[Kephart 2005] J. Kephart, Research Challenges of Autonomic Computing, Proceedings of the International Conference on Software Engineering (ICSE), St. Louis, MO, May 15-21, 2005

[Ksentini 2006] A. Gueroui, M. Naimi, A. Ksentini, Toward an improvement of H.264 video transmission over IEEE 802.11e through a cross-layer architecture, Communications Magazine, IEEE, 44, 1, January, 2006

[Moore 2001] B. Moore, E. Ellesson, J. Strassner, A. Westerinen, Policy Core Information Model -- Version 1 Specification, IETF RFC 3060, February, 2001

[Moore 2003] B. Moore, Policy Core Information Model (PCIM) Extensions, IETF RFC 3460, January, 2003

[Muller 2006] Hausi A. Müller, Liam O'Brien, Mark Klein, Bill Wood, Autonomic Computing, Technical Report - Software Engineering Institute Carnegie Mellon University, April, 2006

[Pena 2006] J. Pena, M. Hinchey, R. Sterritt, A. Ruiz, M. Resinas, A Model-Driven Architecture Approach for Modeling, Specifying and Deploying Policies in Autonomous and Autonomic Systems, Second International Symposium on Dependable Autonomic and Secure Computing (DASC 2006), Indianapolis, Indiana, USA, 29 September - 1 October, 2006

[Smirnov 2004] M. Smirnov, Autonomic Communication: Research Agenda for a New Communications Paradigm, Technical report, Fraunhofer FOKUS, 2004

[Sterritt 2005] R. Sterritt, Autonomic computing, Innovations in Systems and Software Engineering, Nasa Journal, 1, 1, 2005

[Sterritt 2007] R. Sterritt, Systems and software engineering of autonomic and autonomous systems, Innovations in Systems and Software Engineering, Nasa Journal, 3, 1, 2007

[Strassner 2006] J. Kephart, J. Strassner, Autonomic Systems and Networks: Theory and Practice, Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP, Vancouver, BC, 3-7 April, 2006

[Van Wambeke 2007] Nicolas Van Wambeke, Francois Armando, Christophe Chassot, Ernesto Exposito, Architecture and models for self-adaptability of transport protocols, 21st International Conference on Advanced Information Networking and Applications (AINA'2007), Niagara Falls (Canada), 21-23 Mai, 2007

[Van Wambeke 2007b] Nicolas Van Wambeke, Ernesto Exposito, Michel Diaz, Transport layer QoS protocols: the micro-protocol approach, 1st OpenNet Workshop, Bruxelles (Belgium), 27-29 Mars, 2007

[Van Wambeke 2008] Nicolas Van Wambeke, Francois Armando, Christophe Chassot, Ernesto Exposito, A model-based approach for self-adaptive transport protocols, Computer Communications, 31, 11, 2008

[Van Wambeke 2008a] Nicolas Van Wambeke, Francois Armando, Christophe Chassot, Ernesto Exposito, Un modèle de décision pour l'auto-adaptation des protocoles de communication, Colloque Francophone sur l'Ingénierie des Protocoles (CFIP 2008), Les Arcs (France), 25-28 Mars, 2008

[Van Wambeke 2008b] Nicolas Van Wambeke, Ernesto Exposito, Guillaume Jourjon, Emmanuel Lochin, Enhanced transport protocols, End-to-end quality of service over heterogeneous networks, Springer-Verlag New York Inc, 5, 2008

[Van Wambeke 2008c] Nicolas Van Wambeke, Francois Armando, Karim Guennoun, Khalil Drira, Ernesto Exposito, Christophe Chassot, Towards the use of models for autonomic network management, IFIP Joint Conference on Personal Wireless Communications (PWC'2008), Toulouse (France), 30 september - 2 october, 2008

[Van Wambeke 2009] Nicolas Van Wambeke, Autonomic composition of qos oriented communication services, Ph.D. dissertation, Université de Toulouse, Institut National des Sciences Appliquées, Networks and Telecommunications, September 8, 2009

[Van der Schaar 2005] S. Davis, M. Vand der Schaar, Cross-layer wireless multimedia transmission: Challenges, principles and new Paradigms, IEEE Wireless Communications, 12, 4, August, 2005

[Vass 2001] Xinhua Zhuang, S. Zhuang, J. Vass, Scalable, error-resilient, and high-performance video communications in mobile wireless environments, Circuits and Systems for Video Technology, IEEE Transactions on, 11, 7, July, 2001

[Westerinen 2001] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, S. Waldbusser, Terminology for Policy-Based Management, IETF RFC 3198, November, 2001

[White 2004] S. R. White, J. R. Hanson, I. Whalley, D. M. Chess, J. Kephart, An Architectural Approach to Autonomic Computing, 1st International Conference on Autonomic Computing (ICAC'04), IEEE Computer Society Press, New York, May, 2004

Conclusions and perspectives

1. Conclusions

This dissertation has presented our work aimed at supporting the design and the development of the next generation transport layer.

In the first chapter we have presented a state of the art illustrating the current diversity of transport protocols, functions and mechanisms as well as the complexity resulting from the deployment of a particular transport service over the heterogeneous and dynamic services offered by the network layer in order to satisfy the requirements of the vast family of distributed multimedia applications.

We believe that this important evolution of the application and network layers and the increasing diversity of networked devices strongly ask for an innovative and evolutive design of the transport layer architecture. The next generation transport layer must be able to cope with the diversity and complexity involved in the evolving family of transport protocols, functions and mechanisms. Moreover, current and future applications will only be able to take advantage of the most adapted and available transport service if they are able to efficiently interact with the future transport layer.

We have followed a methodology based on a model-driven architecture (MDA) approach aimed at guiding the design of the next generation transport layer. This methodology has been initiated by the analysis of standard transport models and it has been led by the current trends on transport layer evolution.

A first Computation Independent Model (CIM) of the transport layer, providing an abstract and high-level service model and based on the OSI and TCP/IP models has been defined. This CIM model has been enriched by the QoS framework provided by the ITU X.641 recommendation.

Based on this CIM model, a Platform Independent Model (PIM) of the transport layer has been elaborated. The proposed methodology has followed an ontology-driven architecture approach to build the PIM model in the form of a QoS transport ontology in order to provide the required semantic representation integrating standard transport protocols. Moreover, this PIM model provides the required extensibility capabilities aimed at facilitating the integration of future services, functions and mechanisms.

This methodology and the resulting QoS transport ontology model have provided the basis for guaranteeing the required portability, interoperability and reusability capabilities of the next generation transport layer.

In chapter 2, we have presented the requirements guiding the design of the next generation transport layer architecture. These requirements have been expressed from the application programmer and protocol programmer point of views.

From the application programer point of view, the future transport layer needs to provide dynamic services discovery, deployment and binding capabilities in order to facilitate services usability for applications in runtime.

From the transport protocol programmer point of view, this architecture needs to provide an extensible and pluggable framework able to easily integrate new transport components while assuring the interoperability with existing transport protocols. These new components should be easily integrated within a pluggable software architecture allowing new transport services to be automatically discovered, composed and used by the applications in runtime. Moreover, these components could be designed as platform-independent components and during the deployment phase the most adequate platform-specific components could be downloaded and deployed within the transport layer of the networked device.

We have presented our work in designing and developing a component-based transport protocol providing an extensible architecture for composing mechanisms intended to satisfy a large diversity of application requirements over heterogeneous network services.

This transport protocol named the Fully Programmable Transport Protocols (FPTP) offers an unicast, message-oriented, error-controlled, congestion-controlled and time-controlled transport service. Various mechanisms and composition of mechanisms required to provide the adequate FPTP transport functions have been presented and evaluated. The compositional architecture promoted by FPTP has facilitated the specialization and development of new mechanisms aimed at increasing the diversity of the provided transport services.

However, an efficient approach guiding service discovery, composition and deployment should be incorporated within the FPTP architecture, in order to allow current and future applications to dynamically have access to the most adapted FPTP transport services based on the underlying network service.

Based on the lessons learned in developing this protocol, solutions in the area of service-oriented, component-based and semantic-driven architectures have been explored in order to provide guidelines and perspectives for designing the future transport layer architecture.

In chapter 3, our work on designing the adaptive Enhanced Transport Protocol (ETP) has also been presented. ETP has been designed as a specialization of the FPTP protocol and provides an adaptive-oriented architecture facilitating dynamic behavioral or structural adaptation based on the perceived network conditions.

Firstly, behavioral adaptation consists in dynamically tuning transport mechanisms in order to cope with varying network conditions. An interpreter of QoS application-layer semantic has been incorporated within the ETP architecture in order to facilitate the design of behavioral adaptive transport mechanisms. Secondly, structural adaptation consists in replacing one or several of the deployed transport mechanisms (when behavioral adaptation does not offer a satisfactory solution) in order to deploy more adapted mechanisms based on the observed

network conditions. ETP integrates a model-driven approach aimed at guiding both behavioral and structural adaptation of transport mechanisms.

The adaptive composite architecture as well as the behavioral and structural adaptation strategies promoted by ETP provide the required runtime adaptation capabilities in order to face to dynamic network environments.

Moreover, the autonomic computing paradigm has been studied and it appears to be well suited to design self-manage adaptive components guided by the service user policies and the monitored environment conditions.

Based on the lessons learned from ETP and the benefits offered by the autonomic computing paradigm, new studies have been initiated in order to design the Autonomic Transport Protocol (ATP) able to self-manage adaptive transport mechanisms and composite services. ATP should also integrate a knowledge base and a policy framework based on a common semantic model aimed at allowing mechanisms and composite services orchestration.

Figure 1 summarizes our research work and introduces prospective studies aimed at designing the next generation transport layer. The collective research projects participation and thesis involved in this research program are also indicated. My activities related to research projects participation, thesis supervision, scientific production and software implementations are also presented.

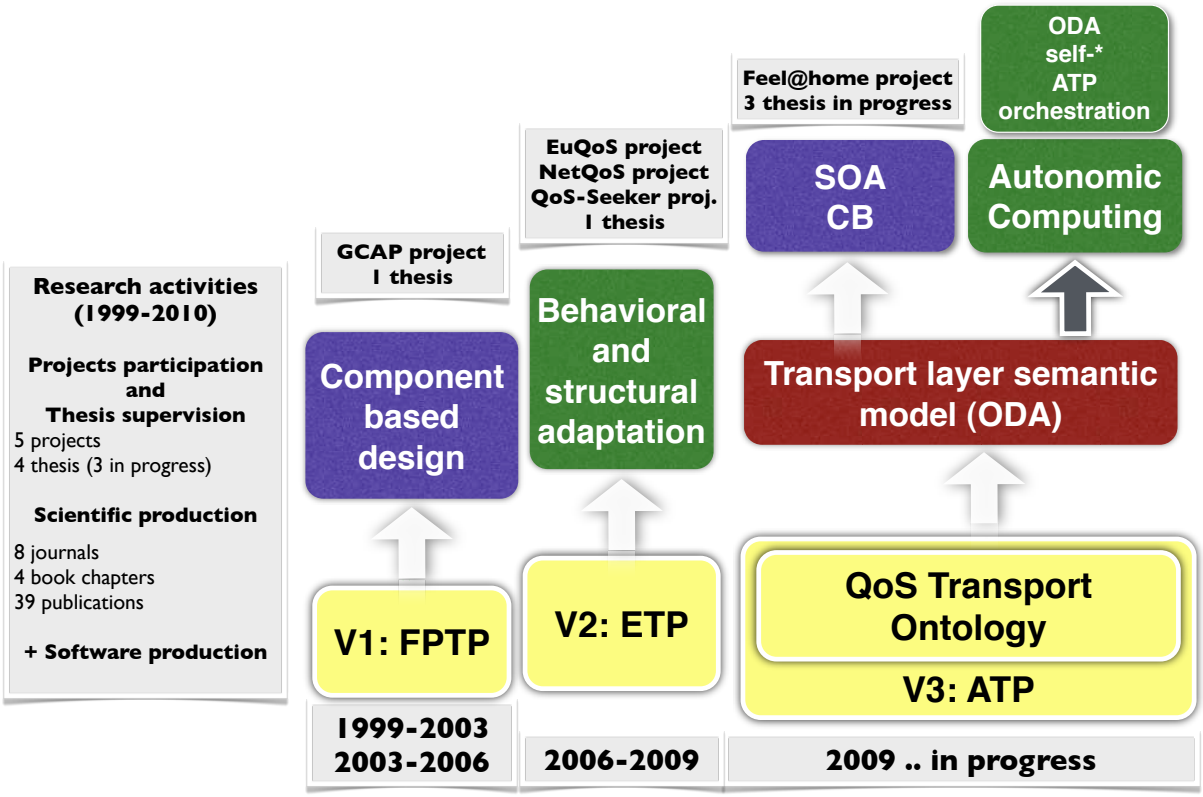

Figure 1. Summary of contributions and perspectives

In the next section, the perspectives of our future work aimed at designing the next generation transport layer able to select, compose, deploy and orchestrate transport protocols following the autonomic computing paradigm and integrating a semantic-driven, component-based and service-oriented approaches will be presented.

2. Perspectives

Our ongoing and future work is intended to design and develop the architecture of the next generation transport layer. This study follows a methodology based on the model-driven architecture approach in order to, firstly define a service-oriented, component-based and semantic-driven architecture and secondly apply the autonomic computing framework to design the future transport layer. The following paragraphs describe this prospective research work.

2.1. Service-oriented architecture

Research studies aimed at designing and developing a service oriented architecture transport layer (SOATL) enabling discovering and selecting the adequate transport service among the large diversity of current and future transport protocols will be carried out.

This architecture should contain a transport services repository accessible from a transport service bus where protocol programmers will be able to publish new transport protocols, functions and mechanisms and application programmers will be able to discover and select the available transport services.

The SOATL architecture will allow efficient and smooth integration and usability of current and future transport protocols.

2.2. Service-component architecture

Studies aimed at designing and developing a generic service-component architecture for transport protocols (SCATP) allowing dynamic composing of transport mechanisms and functions in order to provide the most adequate composite service guided by applications requirements will be effectuated.

Composite transport protocols will consist in a service composite containing a set of pluggable transport mechanisms following an adequate assembly model adapted to the architectural requirements of end-to-end transport protocols. These architectural requirements include integrating data/control and management planes for mechanisms deployment, application and network communication ports as well as adaptive ports for implementing external adaptation strategies.

SCATP will support SOATL by allowing dynamic composition of elementary transport mechanisms (e.g. error detection, error report, rate control, etc) intended to implement the adequate transport functions (e.g. error control, congestion control etc) of composite

transport services (e.g. error-controlled, congestion-controlled and time-controlled tranport service).

2.3. Ontology-driven architecture

Dynamicity, extensibility and heterogeneity properties required by SOATL and SCATP architectures will be guaranteed by the QoS transport ontology model aimed at guiding service selection and composition.

This semantic model will include the required assertions about the available transport mechanisms. These assertions are intended to be processed by reasoning technologies in order to infer the characterization of services offered by transport protocols within SOATL. In this way, SOATL architecture will allow dynamic discovery and selection of transport services. In addition, service-oriented selection based on the expected service properties could be implemented by using the QoS transport ontology. Thus, the selection of these services could be done dynamically based on the specific platform transport protocols available in the user device.

Likewise, the semantic model will contain assertions about fine grained transport mechanisms and their possible compositions aimed at providing a large diversity of transport functions. This component-based semantic information is aimed at guiding the service composition functionalities to be guaranteed by the SCATP architecture.

SOATL and SCATP architectures enriched by the semantic model offered by the QoS transport ontology will allow implementing the required dynamic configuration functionalities for the transport layer of the next generation.

2.4. Autonomic computing framework

Studies aimed at designing an Autonomic Transport Layer (ATL) able to select, compose, deploy and orchestrate transport protocols following the autonomic computing paradigm and integrating the service-oriented, component-based and ontology-drivent architectures will be developed.

The autonomic computing paradigm provides a well suited framework able to integrate the required architecture and the self-managing functionalities of the next generation transport layer.

Studies aimed at designing and developing the Autonomic Transport Protocol (ATP) will be followed up. ATP will conform the autonomic element specification and will be composed of ETP adaptive composites (able to perform behavioral and structural adaption) and specialized autonomic manager (able to self-manage ETP instances).

Moreover, research studies aimed at designing and developing autonomic orchestrators at different levels of the transport services scope will be carried out. These orchestrators will be able to coordinate autonomic elements operating at stream-level (i.e. ATP instances),

application-level (i.e. managers orchestrating multiple streams of the same application), system-level (i.e. managers orchestrating the various application-level manager within the same end-system) and group-level (managers aimed at enforcing adaptation policies between the various end-systems composing a collaborative resources sharing group). This category of autonomic manager orchestrators illustrates the large scope of transport services management and coordination that will be achieved by the next generation autonomic transport layer.

Similarly, studies aimed at enhancing our work in behavioral and structural adaption strategies in order to implement autonomic self-managing functionalities will also be effectuated.

Self-configuring functionalities guided by QoS transport ontology model and the SCATP assembly model will be designed. Semantic-driven and component-based approaches will be followed to provide self-configured autonomic services containing a basic predefined composite (i.e. including basic transport mechanisms such as multiplexing/demultiplexing, segmentation/reassembly, data integrity and connection management), a network-aware composite (i.e. data-control and management functions such as rate control or congestion control taking into account the underlying communication services) and an application-aware composite (i.e. functions composing error control and time control mechanisms). Likewise, an autonomic manager able to measure and analyze changes in the environment and in consequence able to plan and execute behavioral or structural adaptation will be incorporated in the final transport composite service. Moreover, self-configuring functionalities performed by autonomic orchestrators and based on the QoS transport ontology model and the SOATP architecture will also be designed.

Self-optimizing strategies aimed at implementing behavioral adaptation will also be elaborated. Adaptive transport mechanisms able to execute adaptation actions aimed at improving the service offered to the applications while respecting network constraints should be integrated as managed elements of an autonomic transport protocol.

In addition, research studies aimed at enhancing the QoS transport ontology model in order to define a common autonomic knowledge base for self-managing and orchestration of transport services will be carried out. Based on this knowledge base, a common and extensible policy framework enabling both goal and utility-function oriented policies guidance will be designed.

Lastly, future research activities aimed at exploring solutions for self-healing and self-protecting functionalities will be envisaged. Moreover, furthers studies aimed at integrating security, privacy and confidentiality dimensions should also be carried out. Potential research collaborations in this area could be explored to enhance the transport layer architecture design in order to integrate these additional functionalities. Likewise, studies related to collaborative communication middleware services and its applicability in transport services orchestration will be carried out.

Annex: Implementation, validation and inferencing capabilities demonstration

1. QoS transport ontology: implementation, validation and inferencing capabilities demonstration

The QoS transport ontology has been implemented using the OWL language and is available at: ***http://homepages.laas.fr/eexposit/hdr/QoSOntology_V2.owl***

The following paragraphs present various views of the ontology produced by Protege, GrOWL and Swoop tools.

These semantic descriptions have been extracted from the QoS transport ontology presented in the chapter 1.

The consistency of this semantic model (individuals, classes and properties) has been verified using the Pellet reasoner.

Furthermore, the service-oriented and component-based approaches presented in chapter 2 have been integrated by defining axioms related to the various transport mechanisms and functions implemented by the studied transport protocols. In this way, the inferencing capabilities of reasoners have allowed to dynamic validate and produce the semantic QoS characterization of the various transport services (e.g. fully reliable, time-controlled, etc).

Further studies will be carried out in order to integrate axioms related to the influence of mechanisms in the QoS parameters in order to allow dynamic self-adaptation in order to implement a new QoS ontology version aimed at providing the required Autonomic Computing features as presented in chapter 3.

1.1. TCP semantic description

1.1.1. Individual

OWL-Individual: TCP

Instance of:
Transport_protocol
Transport_service
owl:NamedIndividual

Object Assertions:
implements : Segmentation_Reassembly
implements : Connection_management
implements : TCP_congestion_control
implements : Data_integrity
implements : ARQ
implements : Multiplexing_Demultiplexing
implements : TCP_flow_control

1.1.2. Examples of service characterization inferences based on components axioms

TCP rdf:type Fully_reliable:
1) (TCP rdf:type Transport_service)
2) (TCP implements ARQ)
3) |_(ARQ rdf:type Fully_reliable_control_function)
4) |_(Fully_reliable_control_function ⊆ Error_control_function)
5) (Fully_reliable = ((∃implements . Fully_reliable_control_function) ∩ Error_controlled_service))
6) |_(Error_controlled_service = (Transport_service ∩ (∃implements . Error_control_function)))

TCP rdf:type Fully_ordered:
1) (TCP rdf:type Transport_service)
2) (TCP implements ARQ)
3) |_(ARQ rdf:type Fully_ordered_control_function)
4) |_(ARQ rdf:type Fully_reliable_control_function)
5) |_(Fully_reliable_control_function ⊆ Error_control_function)
6) (Error_controlled_service = (Transport_service ∩ (∃implements . Error_control_function)))
7) (Fully_ordered = (Error_controlled_service ∩ (∃implements . Fully_ordered_control_function)))

TCP rdf:type Flow_controlled:
1) (TCP rdf:type Transport_service)
2) (TCP implements TCP_flow_control)
3) |_(TCP_flow_control rdf:type Flow_control_function)
4) |_(Flow_control_function ⊆ Throughput_control_function)
5) (Flow_controlled = ((∃implements . Flow_control_function) ∩ Throughput_controlled_service))
6) |_(Throughput_controlled_service = (Transport_service ∩ (∃implements . Throughput_control_function)))

TCP rdf:type Congestion_controlled:
1) (TCP rdf:type Transport_service)
2) (TCP implements TCP_congestion_control)
3) |_(TCP_congestion_control rdf:type Congestion_control_function)
4) |_(Congestion_control_function ⊆ Throughput_control_function)
5) (Congestion_controlled = ((∃implements . Congestion_control_function) ∩ Throughput_controlled_service))
6) |_(Throughput_controlled_service = (Transport_service ∩ (∃implements . Throughput_control_function)))

TCP rdf:type Error_and_throughput_controlled_service:
1) (TCP rdf:type Transport_service)
2) (TCP implements ARQ)
3) |_(TCP implements TCP_flow_control)
4) |_(TCP_flow_control rdf:type Flow_control_function)
5) |_(Flow_control_function ⊆ Throughput_control_function)
6) |_(ARQ rdf:type Fully_reliable_control_function)
7) |_(Fully_reliable_control_function ⊆ Error_control_function)
8) (Error_controlled_service = (Transport_service ∩ (∃implements . Error_control_function)))
9) (Error_and_throughput_controlled_service = (Error_controlled_service ∩ Throughput_controlled_service))
10) |_(Throughput_controlled_service = (Transport_service ∩ (∃implements . Throughput_control_function)))

1.2. UDP semantic description

1.2.1. Individual

OWL-Individual: UDP

Instance of:
Transport_protocol
Transport_service
owl:NamedIndividual

Object Assertions:
implements : Data_integrity
implements : Multiplexing_Demultiplexing

1.3. SCTP semantic description

1.3.1. Individuals (standard SCTP and PR-SCTP)

OWL-Individual: SCTP

Instance of:
Transport_protocol
Transport_service
owl:NamedIndividual

Object Assertions:
implements : TCP-like_congestion_control
implements : Segmentation_Reassembly
implements : Multiple_stream_management
implements : Connection_management
implements : ARQ
implements : Data_integrity
implements : Multiplexing_Demultiplexing
implements : TCP_flow_control

OWL-Individual: PR-SCTP

Instance of:
Transport_protocol
Transport_service
owl:NamedIndividual

Object Assertions:
implements : TCP-like_congestion_control
implements : Segmentation_Reassembly
implements : Multiple_stream_management
implements : PR
implements : Connection_management
implements : Data_integrity
implements : Multiplexing_Demultiplexing
implements : TCP_flow_control

1.3.2. Examples of service characterization inferences based on components axioms

SCTP rdf:type Fully_reliable:
1) (SCTP implements ARQ)
2) |_(ARQ rdf:type Fully_reliable_control_function)
3) |_(Fully_reliable_control_function ⊆ Error_control_function)
4) (SCTP rdf:type Transport_service)
5) (Fully_reliable ≡ (Error_controlled_service ∩ (∃implements . Fully_reliable_control_function)))
6) |_(Error_controlled_service ≡ ((∃implements . Error_control_function) ∩ Transport_service))

PR-SCTP rdf:type Partially_reliable:
1) (PR-SCTP implements PR)
2) |_(PR-SCTP implements Multiple_stream_management)
3) |_(PR-SCTP rdf:type Transport_service)
4) |_(Multiple_stream_management rdf:type Partially_ordered_control_function)
5) |_(Partially_ordered_control_function ⊆ Error_control_function)
6) |_(PR rdf:type Partially_reliable_control_function)
7) (Partially_reliable ≡ ((∃implements . Partially_reliable_control_function) ∩ Error_controlled_service))
8) |_(Error_controlled_service ≡ ((∃implements . Error_control_function) ∩ Transport_service))

SCTP rdf:type Fully_ordered:
1) (SCTP implements ARQ)
2) |_(ARQ rdf:type Fully_ordered_control_function)
3) |_(Fully_ordered_control_function ⊆ Error_control_function)
4) (SCTP rdf:type Transport_service)
5) (Fully_ordered ≡ ((∃implements . Fully_ordered_control_function) ∩ Error_controlled_service))
6) |_(Error_controlled_service ≡ ((∃implements . Error_control_function) ∩ Transport_service))

SCTP rdf:type Partially_ordered:
1) (SCTP implements Multiple_stream_management)
2) |_(Multiple_stream_management rdf:type Partially_ordered_control_function)
3) |_(Partially_ordered_control_function ⊆ Error_control_function)
4) (SCTP rdf:type Transport_service)
5) (Partially_ordered ≡ ((∃implements . Partially_ordered_control_function) ∩ Error_controlled_service))
6) |_(Error_controlled_service ≡ ((∃implements . Error_control_function) ∩ Transport_service))

SCTP rdf:type Congestion_controlled:

1) (SCTP implements TCP-like_congestion_control)
2) |_(TCP-like_congestion_control rdf:type Congestion_control_function)
3) |_(Congestion_control_function ⊆ Throughput_control_function)
4) (SCTP rdf:type Transport_service)
5) (Congestion_controlled = ((∃implements . Congestion_control_function) ∩ Throughput_controlled_service))
6) |_(Throughput_controlled_service = (Transport_service ∩ (∃implements . Throughput_control_function)))

SCTP rdf:type Flow_controlled:

1) (SCTP implements TCP_flow_control)
2) |_(SCTP implements TCP-like_congestion_control)
3) |_(SCTP rdf:type Transport_service)
4) |_(TCP-like_congestion_control rdf:type Congestion_control_function)
5) |_(Congestion_control_function ⊆ Throughput_control_function)
6) |_(TCP_flow_control rdf:type Flow_control_function)
7) (Flow_controlled = ((∃implements . Flow_control_function) ∩ Throughput_controlled_service))
8) |_(Throughput_controlled_service = (Transport_service ∩ (∃implements . Throughput_control_function)))

SCTP rdf:type Error_and_throughput_controlled_service:

1) (SCTP implements TCP-like_congestion_control)
2) |_(SCTP implements ARQ)
3) |_(SCTP rdf:type Transport_service)
4) |_(ARQ rdf:type Fully_ordered_control_function)
5) |_(Fully_ordered_control_function ⊆ Error_control_function)
6) |_(TCP-like_congestion_control rdf:type Congestion_control_function)
7) |_(Congestion_control_function ⊆ Throughput_control_function)
8) (Error_and_throughput_controlled_service = (Error_controlled_service ∩ Throughput_controlled_service))
9) |_(Error_controlled_service = ((∃implements . Error_control_function) ∩ Transport_service))
10) |_(Throughput_controlled_service = (Transport_service ∩ (∃implements . Throughput_control_function)))

1.4. DCCP semantic description

1.4.1. Individuals (DCCP-2, DCCP-3 and DCCP-4 profiles)

OWL-Individual: DCCP-2

Instance of:
Transport_protocol
Transport_service
owl:NamedIndividual

Object Assertions:
implements : TCP-like_congestion_control
implements : Data_integrity
implements : Multiplexing_Demultiplexing

OWL-Individual: DCCP-3

Instance of:
Transport_protocol
Transport_service
owl:NamedIndividual

Object Assertions:
implements : Data_integrity
implements : Multiplexing_Demultiplexing
implements : TFRC_congestion_control

OWL-Individual: DCCP-4

Instance of:
Transport_protocol
Transport_service
owl:NamedIndividual

Object Assertions:
implements : Data_integrity
implements : TFRC-SP_congestion_control
implements : Multiplexing_Demultiplexing

1.4.2. Examples of service characterization inferences based on components axioms

DCCP-2 rdf:type Congestion_controlled:
1) (DCCP-2 rdf:type Transport_service)
2) (DCCP-2 implements TCP-like_congestion_control)
3) |_(TCP-like_congestion_control rdf:type Congestion_control_function)
4) |_(Congestion_control_function ⊆ Throughput_control_function)
5) (Congestion_controlled ≡ ((∃implements . Congestion_control_function) ∩ Throughput_controlled_service))
6) |_(Throughput_controlled_service ≡ (Transport_service ∩ (∃implements . Throughput_control_function)))

DCCP-3 rdf:type Congestion_controlled:
1) (DCCP-3 rdf:type Transport_service)
2) (DCCP-3 implements TFRC_congestion_control)
3) |_(TFRC_congestion_control rdf:type Congestion_control_function)
4) |_(Congestion_control_function ⊆ Throughput_control_function)
5) (Congestion_controlled ≡ ((∃implements . Congestion_control_function) ⊓ Throughput_controlled_service))
6) |_(Throughput_controlled_service ≡ (Transport_service ⊓ (∃implements . Throughput_control_function)))

DCCP-4 rdf:type Congestion_controlled:
1) (DCCP-4 rdf:type Transport_service)
2) (DCCP-4 implements TFRC-SP_congestion_control)
3) |_(TFRC-SP_congestion_control rdf:type Congestion_control_function)
4) |_(Congestion_control_function ⊆ Throughput_control_function)
5) (Congestion_controlled ≡ ((∃implements . Congestion_control_function) ⊓ Throughput_controlled_service))
6) |_(Throughput_controlled_service ≡ (Transport_service ⊓ (∃implements . Throughput_control_function)))

1.5. MPTCP semantic description

1.5.1. Individual

OWL-Individual: MPTCP

Instance of:
Transport_protocol
Transport_service
owl:NamedIndividual

Object Assertions:
implements : Segmentation_Reassembly
implements : TCP_congestion_control
implements : Connection_management
implements : Data_integrity
implements : ARQ
implements : Multipath_management
implements : Multiplexing_Demultiplexing
implements : TCP_flow_control

1.5.2. Examples of service characterization inferences based on components axioms

MPTCP rdf:type Fully_reliable:
1) (MPTCP implements ARQ)
2) |_(ARQ rdf:type Fully_reliable_control_function)
3) |_(Fully_reliable_control_function ⊆ Error_control_function)
4) (MPTCP rdf:type Transport_service)
5) (Fully_reliable ≡ (Error_controlled_service ⊓ (∃implements . Fully_reliable_control_function)))
6) |_(Error_controlled_service ≡ ((∃implements . Error_control_function) ⊓ Transport_service))

MPTCP rdf:type Fully_ordered:
1) (MPTCP implements ARQ)
2) |_(ARQ rdf:type Fully_ordered_control_function)
3) |_(Fully_ordered_control_function ⊆ Error_control_function)
4) (MPTCP rdf:type Transport_service)
5) (Fully_ordered ≡ ((∃implements . Fully_ordered_control_function) ⊓ Error_controlled_service))
6) |_(Error_controlled_service ≡ ((∃implements . Error_control_function) ⊓ Transport_service))

MPTCP rdf:type Congestion_controlled:
1) (MPTCP implements TCP_congestion_control)
2) |_(TCP_congestion_control rdf:type Congestion_control_function)
3) |_(Congestion_control_function ⊆ Throughput_control_function)
4) (MPTCP rdf:type Transport_service)
5) (Congestion_controlled ≡ ((∃implements . Congestion_control_function) ⊓ Throughput_controlled_service))
6) |_(Throughput_controlled_service ≡ (Transport_service ⊓ (∃implements . Throughput_control_function)))

MPTCP rdf:type Flow_controlled:
1) (MPTCP implements TCP_flow_control)
2) |_(TCP_flow_control rdf:type Flow_control_function)
3) |_(Flow_control_function ⊆ Throughput_control_function)
4) (MPTCP rdf:type Transport_service)
5) (Flow_controlled ≡ ((∃implements . Flow_control_function) ⊓ Throughput_controlled_service))
6) |_(Throughput_controlled_service ≡ (Transport_service ⊓ (∃implements . Throughput_control_function)))

MPTCP rdf:type Error_and_throughput_controlled_service:
1) (MPTCP implements TCP_flow_control)
2) |_(MPTCP implements ARQ)
3) |_(MPTCP rdf:type Transport_service)
4) |_(ARQ rdf:type Fully_ordered_control_function)
5) |_(Fully_ordered_control_function ⊆ Error_control_function)
6) |_(TCP_flow_control rdf:type Flow_control_function)
7) |_(Flow_control_function ⊆ Throughput_control_function)
8) (Error_and_throughput_controlled_service ≡ (Error_controlled_service ⊓ Throughput_controlled_service))
9) |_(Error_controlled_service ≡ ((∃implements . Error_control_function) ⊓ Transport_service))
10) |_(Throughput_controlled_service ≡ (Transport_service ⊓ (∃implements . Throughput_control_function)))

1.6. FPTP semantic description

1.6.1. Individual

OWL-Individual: FPTP

Instance of:
Transport_service
Composite_transport_protocol
owl:NamedIndividual

Object Assertions:
implements : FPTP_TD_RC
implements : FPTP_D_PR
implements : FPTP_loss_detection
implements : FPTP_TD_PR
implements : FPTP_partial_retransmission
implements : FPTP_no_feedback
implements : FPTP_TFRC
implements : FPTP
implements : FPTP_RC
implements : FPTP_feedback
implements : FPTP_create_feedback
implements : FPTP_time_constrained_retransmission
implements : FPTP_qos_parser
implements : FPTP_TD_TFRC
implements : FPTP_PR
implements : FPTP_differentiated_retransmission
implements : FPTP_retransmission
implements : FPTP_FR
implements : FPTP_T_PR
implements : FPTP_time_constrained_differentiated_retransmission

1.6.2. Examples of service characterization inferences based on components axioms

FPTP rdf:type Fully_reliable:
1) (FPTP rdf:type Transport_service)
2) (FPTP implements FPTP_FR)
3) |_(FPTP_FR rdf:type Fully_reliable_control_function)
4) |_(Fully_reliable_control_function ⊆ Error_control_function)
5) (Fully_reliable = (Error_controlled_service ∩ (∃implements . Fully_reliable_control_function)))
6) |_(Error_controlled_service = ((∃implements . Error_control_function) ∩ Transport_service))

FPTP rdf:type Partially_reliable:
1) (FPTP implements FPTP_TD_PR)
2) |_(FPTP_TD_PR rdf:type Partially_reliable_control_function)
3) |_(Partially_reliable_control_function ⊆ Error_control_function)
4) (FPTP rdf:type Transport_service)
5) (Partially_reliable = ((∃implements . Partially_reliable_control_function) ∩ Error_controlled_service))
6) |_(Error_controlled_service = ((∃implements . Error_control_function) ∩ Transport_service))

FPTP rdf:type Fully_ordered:
1) (FPTP rdf:type Transport_service)
2) (FPTP implements FPTP_FR)
3) |_(FPTP_FR rdf:type Fully_ordered_control_function)
4) |_(Fully_ordered_control_function ⊆ Error_control_function)
5) (Fully_ordered = ((∃implements . Fully_ordered_control_function) ∩ Error_controlled_service))
6) |_(Error_controlled_service = ((∃implements . Error_control_function) ∩ Transport_service))

FPTP rdf:type Rate_controlled:
1) (FPTP implements FPTP_TD_TFRC)
2) |_(FPTP implements FPTP_RC)
3) |_(FPTP rdf:type Transport_service)
4) |_(FPTP_RC rdf:type Rate_control_function)
5) |_(FPTP_TD_TFRC rdf:type Congestion_control_function)
6) |_(Congestion_control_function ⊆ Throughput_control_function)
7) (Throughput_controlled_service = ((∃implements . Throughput_control_function) ∩ Transport_service))
8) (Rate_controlled = ((∃implements . Rate_control_function) ∩ Throughput_controlled_service))

FPTP rdf:type Congestion_controlled:
1) (FPTP implements FPTP_TD_TFRC)
2) |_(FPTP_TD_TFRC rdf:type Congestion_control_function)
3) |_(Congestion_control_function ⊆ Throughput_control_function)
4) (FPTP rdf:type Transport_service)
5) (Throughput_controlled_service = ((∃implements . Throughput_control_function) ∩ Transport_service))
6) (Congestion_controlled = ((∃implements . Congestion_control_function) ∩ Throughput_controlled_service))

FPTP rdf:type Error_throughput_and_time_controlled_service:
1) (FPTP implements FPTP_FR)
2) |_(FPTP implements FPTP_TD_TFRC)
3) |_(FPTP rdf:type Transport_service)
4) |_(FPTP implements FPTP_T_PR)
5) |_(FPTP_T_PR rdf:type Time_control_function)
6) |_(FPTP_TD_TFRC rdf:type Congestion_control_function)
7) |_(Congestion_control_function ⊆ Throughput_control_function)
8) |_(FPTP_FR rdf:type Fully_ordered_control_function)
9) |_(Fully_ordered_control_function ⊆ Error_control_function)
10) (Time_controlled_service = (Transport_service ∩ (∃implements . Time_control_function)))
11) (Error_controlled_service = (Transport_service ∩ (∃implements . Error_control_function)))
12) (Error_throughput_and_time_controlled_service = (Time_controlled_service ∩ Error_controlled_service ∩ Throughput_controlled_service))
13) |_(Throughput_controlled_service = ((∃implements . Throughput_control_function) ∩ Transport_service))