Project Number:	**IST-1999-10077**
Project Title:	**AQUILA** **Adaptive Resource Control for QoS Using an IP-based Layered Architecture**
Deliverable Type:	**PU – public**

Deliverable Number:	**IST-1999-10077-WP1.2-SAG-1202-PU-O/b1**
Contractual Date of Delivery to the CEC:	**September 30, 2001**
Actual Date of Delivery to the CEC:	**September 28, 2001: version b0** **December 31, 2001: version b1**
Title of Deliverable:	**System architecture and specification for the second trial**
Workpackage contributing to the Deliverable:	**WP 1.2**
Nature of the Deliverable:	**O – Other (Specification)**
Editor:	**Martin Winter (SAG)**
Author(s):	**Estíbaliz Bear (SAG); Lila Dimopoulou (NTU); Heinz Dörken (DTA); Gerald Eichler (DTA); Thomas Engel (SAG); Reinhard Frank (SAG); Falk Fünfstück (TUD); Axel Hof (SAG); John Karadimas (QSY); Dietmar Katzengruber (TAA); Sotiris Maniatis (NTU); Natalia Miettinen (ELI); Eugenia Nikolouzou (NTU); Fabio Ricciato (COR); Stefano Salsano (COR); Petros Sampatakos (NTU); Anne Thomas (TUD); G. Troiano (COR); Haris Tsetsekas (NTU); Ralf Widera (DTA); Martin Winter (SAG)**

Abstract:	This deliverable D1202 specifies the architecture for the second trial.
Keyword List:	AQUILA, IST, architecture, resource control layer, QoS

Executive Summary

This deliverable describes the system architecture used for the prototypes in the second trial. Additionally, studies on related topics, which are not going to be implemented, are included in this document. The base for this specification is D1201. For the second trial, a couple of extensions are specified, including

- Inter-operation with MPLS
- Enhanced resource distribution, taking into account extended network topologies
- Feedback of network usage information into several points of the resource control process
- Extended application interface for the support of legacy application, SIP inter-working and grouping of a set of related reservations
- Inter-domain resource reservations
- Logging of usage data
- Extended QoS management capabilities

Studies are provided covering

- Advance reservation
- Content delivery networks
- IP multicast
- Security issues

Table of Contents

Table of Figures

Table of Tables

1 Introduction

1.1 Architectural approach

The AQUILA project tackles with the project targets in a two-phased approach:

- In the first phase, basic mechanisms for resource control and traffic handling were developed, implemented and tested. A general framework for application interfaces was specified and the basic functions have been implemented and used for the first trial. The result of this first phase is a QoS architecture for a single domain, which meets the requirements of the project.
- The second phase is based on this architecture and on the results and experiences of the first trial. It will extend the project in the following ways:
    - Enhance and modify the mechanisms already developed according to the results of the first trial
    - Add additional functionality to cover a broader range of applicability, e.g. for inter-domain resource allocation or by adding feedback from the network.
    - Perform studies on related topics, which are important for QoS provisioning, but will not be implemented in the trial.

The main innovations of the second phase can be summarised as follows:

- Added support for a larger variety of network topologies and technologies.
- Scalable resource reservations over multiple domains.
- Control loops at various points of the architecture, to feed back network usage information into the admission control, resource distribution and provisioning process.
- Wider support for applications, including an enhanced legacy application support and a SIP signalling gateway.
- Enhanced management of QoS features.

This document, which specifies the architecture for the second trial, follows these general ideas. Because extensions and modifications of the architecture are taking place at various points of the architecture, the reader may just pick the chapters he/she is interested in and skip others. Especially chapters 5 and 6 may not be of general interest, but are intended for the people, which are going to design the components of the resource control layer.

1.2 Document structure

The document consists of four parts:

- Part I analyses the results of the first trial, determines the requirements and sketches the approach used for the second trial. It consists of a single chapter 2, containing a general description of the extended architecture, taking into account the vision of the project and the results already achieved in the first trial. These two corners define the approach of the project for the second trial. Additionally, this part contains a definition of important terms.
- Part II contains the detailed architectural analysis and studies. It is divided into two chapters:
  - Chapter 3 specifies the overall structure and architecture, divided into several focal issues: Network architecture, resource control layer, inter-domain resource allocation and application interfaces.
  - Chapter 4 contains studies on related topics, which are not going to be implemented, but are considered to be important in relation to the project.
- Part III provides a coarse specification of the components and thus builds the bridge to the implementation work done in work packages 2.1 and 2.2 and provides the base for the design specification D2103.
  - Chapter 5 is a coarse specification of the system components.
  - Chapter 6 specifies the interfaces between the various functional units.
- Part IV contains supporting chapters
  - Chapter 7 is the list of abbreviations used.
  - Chapter 8 finally lists the references.

Part I: Requirements and approach

2 Approach and definitions

2.1 Approach

2.1.1 Outcomes of the first trial

The principles of the architecture of the AQUILA's Resource Control Layer (RCL), as described in [D1201], have been validated by the first trial experiments. In the following a short summary of the first trial results and a feedback for the second trial is given.

The first trial two main objectives were: the verification of the network service implementation and the evaluation of RCL performance [D3201]. Whereas the first objective addresses mainly traffic engineering issues [D1302], the results of the second directly influence the architecture for the second trial described in this document.

Generally, the network services, their implementation as well as the RCL components and algorithms software fulfilled the expectations of the first trial. However, there are some conclusions regarding the RCL, which have to be considered by specifying the second trial architecture and the algorithms:

- The resource pools algorithms should be refined. Particularly, the traffic class 2, which implements the network service PVBR, showed some differences to the specification in [D1301].
- The configuration of the resource pools exposed as a difficult task. First, it is an open problem how to configure the traffic classes in the resource pool hierarchy. Second, the meaning of the TrafficSpec used for the configuration in context of resource pools is not clear. There might be too many and too detailed parameters.
- The performance of the RCL should be improved. The RCL components produce a significant time-consuming signalling load, particularly at start-up as well as for reservation requests and releases, which probably affect realistic QoS scenarios. For example, the high signalling load for a reservation initiation is mainly caused by the ACA logging, the router and the database access.
- The first trial identified the possibility for control loops between the RCL and the Measurement Toolkit. Measured parameters may dynamically influence the decisions taken by the RCL, for example by the admission control. The admission control, so far, relied only on statistic methods such as the resource distribution via the resource pools.

The RCL architecture for the first trial, moreover, focused on basic functionality in order to quickly develop an RCL prototype and get early trial results. Consequently, there were some major restrictions as shown in the following. It is another consequence for the second phase to develop an architecture that theoretically considers the below listed open issues, and practically realises the most important ones:

- The first trial architecture assumed a single domain scenario; the RCL handled intra-domain issues only. Inter-domain issues will first time be considered in the second trial.
- The for the first trial chosen network topology is rather simple and does not consider many characteristics of large provider networks, such as network redundancy, multiple homing, and complex access networks. However, these are of interest for the second trial.
- The integration of MPLS in the AQUILA approach was also not focus of the first trial. This document for the second trial outlines possible scenarios on how to support MPLS and VPN.
- In the first trial, only unidirectional, single QoS reservations were possible. The second trial approach, however, allows the creation of reservation groups.
- QoS reservations for IP multicast sessions (reservation style: p2m) were not taken into consideration as well. The second trial architecture makes some proposals to support QoS for multicast.
- The logging of usage data, useful for charging and accounting, was foreseen but not implemented by the first trial prototype. It will be realised by the second one.
- The use of signalling protocols for QoS requests was not implemented yet. SIP and RSVP can be an alternative way to reserve QoS resources within the RCL. The Proxy for the second trial will practically realise a protocol gateway for SIP. Moreover, proposals are made how to support RSVP.
- The API for QoS-aware applications was a very early, AQUILA proprietary version. However, it is going to be refined and extended for the second trial purposes. Furthermore, the API is open for generalisation as described in this document.
- The QMTool realised also a very basic functionality for the first phase, and will be enhanced with additional management functions for the second one.
- The first trial prototype considered security issues only at user identification level. There is a need for studies, how security could be integrated in here specified approach.

2.1.2 The approach for the second trial

The following chapter gives an overview of the architectural approach for the second trial. Basically, this architecture is based on the specification made for the first trial [D1201]. However, a number of new features, enhancements, and refinements are integrated. Their introduction is the focus of this chapter where as detailed specifications follow in the chapters 3 and 4 (analysis), and 5 and 6 (design).

The architecture for the second trial extends and refines the one, which was realised for the first trial. It relies on the Resource Control Layer that acts as distributed bandwidth broker [RFC2638] and which controls the resources of the underlying DiffServ-aware network. The approach for the second trial is however not longer restricted to a single domain scenario. It is also able to control resources between different domains that are not necessarily all controlled by AQUILA Resource Control Agents.

The Resource Control Layer for the second trial basically consists of the same components (agents) as described in [D1201]:

- The **Resource Control Agent (RCA)**, to control the whole resources of one domain and to distribute them by using a resource pool tree.

- The **Admission Control Agent (ACA)**, to act as leaf of the tree and to perform local admission and policy control.

- The **End-user Application Toolkit (EAT)**, to reside at the edge between the end-user, the applications and the RCL in order to show the offered network services and to perform QoS reservation requests.

The above listed components take care of the control of one domain. To consider inter-domain issues, a new component is introduced:

- The **BGRP Agent**, to communicate reservation requests between domains that are locally controlled by RCAs, for example. It interacts with the egress and ingress ACAs, respectively, of the neighbour domains.

All together form a two-levelled logical overlay network above the underlying core network. While the lower level is mainly identical to the first trial approach and restricted to intra-domain issues, the new second level deals with inter-domain issues [Figure 2-1].

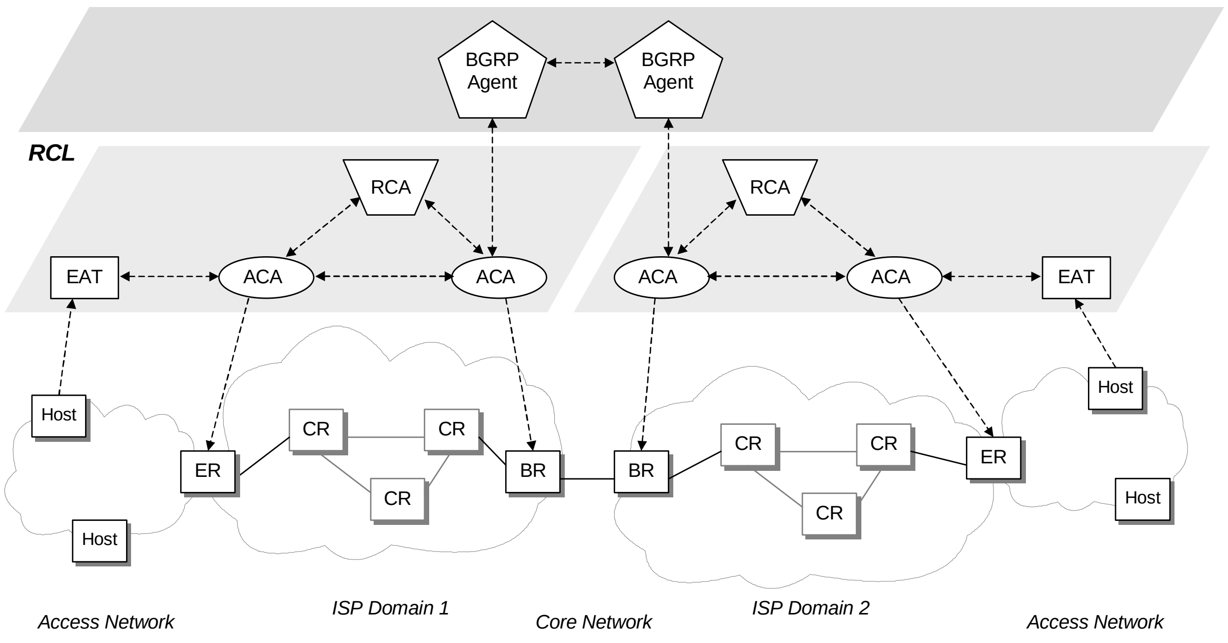

Figure 2-1: The architecture of the RCL for the second trial

The figure above roughly depicts the main components of the RCL and their communication interfaces.

With respect to this most important innovation, there are further essential changes to the first trial:

Topics to be realised

The following paragraphs shortly introduce the new topics of the architecture of the second trial, which are going to be *implemented* by the project. They build a summary of the add-ons to the first trial architecture which is still the base for the in this document described approach.

2.1.2.1 Resource distribution & pools

In AQUILA, resources are distributed by using the resource pool approach, which is going to be adapted for the needs of the second trial. First of all, the performance can be improved by reducing the resource parameter set to only *one* parameter: the *bandwidth*. That also makes the configuration of traffic classes in the hierarchical manner of the resource pool tree easier.

In general, resource pools are configured in two phases: the tree creation phase based on static rules (initial resource provisioning), and the dynamic re-distribution phase. Latter may take place when an ACA asks for more resources, for example to admit a reservation request that exceeds its resource limit. On the other hand, resources may be returned if the sum of all reserved resources goes under a *low watermark*.

Resource pools are still at intra-domain level, and the RCA is the component, which takes care of the creation of them. Inter-domain issues are at BGRP level and handled there.

More details can be found in the chapters 3.1.1 and 5.2

2.1.2.2 Inter-domain resource control

Resource control between domains has some major differences to resource control within a domain. This enforces the need to establish new mechanisms to handle the inter-domain case.

- A domain's topology is built and controlled by a single network operator. Therefore, abstraction mechanisms like the resource pools can successfully be used to provide a coarse view of the topology. The Internet topology however, is based on a set of bilateral agreements between network operators.

- Scalability is an issue both for intra-domain and inter-domain resource control. One can handle single flows within a domain with an approach using distributed processing (one ACA per edge device). For a scalable inter-domain resource control however it is necessary to aggregate flows [BGRP].

Taking into account these differences, the project specifies and develops a different resource control mechanism used between domains, based on the proposal made in [BGRP]. A hop-by-hop reservation, based on the BGP routes, is used to follow the bilateral agreements of neighboured domains. Sink tree based aggregation of reservations is the base for scalability. Early reservation responses ("quiet grafting") are used to control the amount of signalling traffic. Open interfaces allow this reservation mechanism to be used in conjunction with any kind of intra-domain resource control, which can provide edge-to-edge QoS.

The project proposes a set of globally well known services to provide a common understanding of network services and to allow the creation of Internet-wide services.

More details can be found in the chapters 3.2 and 5.3.

2.1.2.3 Network topology

The first trial was restricted to a relatively simple network scenario. The architecture for the second trial will consider – with regard to the practicability – more characteristic scenarios:

- Dual Hosting, where one host is connected to many edge routers. This influences the reservations, particularly the IP addresses.

- Dual Homing, where one edge router is connected to many core routers. The project has particularly to take care of situations where the links are not controlled by the same higher level pool.

- Dual Peering Points, where two autonomous systems (domains) are connected to many border routers. This issue has to be considered by the inter-domain resource control.

More details can be found in the chapter 3.1.2.

2.1.2.4 Control loops

The RCL for the first trial used a rather conservative mechanism to distribute resources and admit resource requests. In this so-called "open loop control", no feedback from the network is considered but mainly static mechanisms and rules. It is therefore task of the second trial approach to *close* the control loop by taking measured network parameters, such as utilisation, delay, etc., into account.

In AQUILA, control loops are relevant for:

- The measurement-based admission control (MBAC). At the (ingress or egress) router, passively measured (monitored) statistical parameters influence the decisions taken by the ACA in order to admit more requests and better utilise resources.

- The re-provisioning of resources. By using some feedback from the network, a more dynamic reconfiguration of the resource pools is possible, e.g. by adapting pool values, or by changing WFQ parameters in a longer time scale instead of directly shifting resources between the traffic classes.

More details about measurement-based admission control can be found in the chapters 3.1.3 and 5.1.1. Control loops for re-provisioning are detailed in chapter 4.5. The level of implementation of the latter depends on some still unsolved issues regarding the possibility to adapt WFQ parameters in running and deployed networks.

2.1.2.5 Reservation groups

With the new architecture it will be possible to group several reservation requests and send them together as a single request. The features are:

- Bi-directional services can be supported. Instead of requesting two independent, uni-directional reservations, a reservation group is formed containing single reservations for both directions.

- Larger reservation groups will also be supported, for example for multi-conference sessions. The end-user, moreover, will have the possibility to join or to quit a former established group.

- Often, a reservation group shall only be established if all single reservations can be accepted. In this case, the request interface contains a flag "complete_group_needed".

- The EAT will be able to offer reservation groups to end-user in a comfortable way. For example, only one reservation form for a bi-directional service has to be filled. The EAT will then internally form the group with the two single reservations and submit it to the ACA.

More details can be found in the chapters 3.1.4 and 5.1.2.

2.1.2.6 Prioritised signalling

One of the outcomes of the first trial concerning the RCL performance is that the RCL in fact produces a significant signalling overhead. Signalling traffic occurs within the RCL (between the RCL components and the database) as well as between the RCL and external entities (such as end-users, applications, and DiffServ-aware routers).

Particularly at start-up, for reservation requests, database and router access, the RCL produces a sizeable message transfer [D3201]. This control traffic is, however, submitted by using the same physical links as the user traffic. It is thus important to prioritise the signalling traffic of the RCL in order to avoid situations where the user traffic affects the control traffic. Existing services for transaction oriented applications might be reused for this purpose. Different solutions are proposed on how to use DSCPs for signalling traffic.

More details can be found in the chapters 3.1.5 and 5.6.1.

2.1.2.7 Logging of usage data

The project provides functions for the *logging* and *accounting* of usage data; charging and billing are out of the focus.

The responsible component for the logging of usage data is the ACA. It collects the data from the router's MIB and provides them for further purposes in two ways:

- For the end-user, accounting data are provided for each reservation (on per flow basis). The EAT can retrieve these data through the ACA's interface and present them to the end-user.

- For the provider (ISP), the usage data are provided as statistic file, accessible for further processing for charging and billing purposes.

More details can be found in the chapters 3.1.6 and 5.1.3.

2.1.2.8 End-user application support

It is also the task of the second trial to extend and improve the support of end-user applications. This will be done in a threefold way:

- For legacy applications that are not QoS-aware, the former Usual Reservation GUI of the first trial is a prototype that shows also, how the integration of QoS offers in a Complex Internet Services (CIS) can be realised in a user-friendly way. The Application Profiles and the EAT's Converters will be refined in a way that such services can easily be completed with the possibility to present network's QoS options and to allow reservation requests.

- The legacy application support actually uses the internal EAT API. This API will be refined and extended to support reservation groups, for example. Moreover, a new, more general API is discussed in this paper which aims at the provision of high-level QoS functions for the development of new, QoS-aware applications. The aim is not to present another proprietary API but to show how a generic QoS provision could be realised, and how this new API inter-works with lower-level APIs such as the EAT API.

- For applications that are not EAT-based, additional protocol gateways (here also called: Proxies) will be realised. However, as an enhancement to the first trial, these proxies will not only be able to detect flow information for the completion of reservation requests, but will directly translate QoS requests via signalling protocols into RCL-conform requests. So applications that use SIP or RSVP to signal their QoS requirements can benefit from the RCL QoS capabilities as well.

More details can be found in chapters the 3.3, 5.4 and 6.3.

2.1.2.9 Management

The QMTool, which was already realised for the first trial, is going to be updated and extended for the second trial. It will be able to:

- design the resource pool tree in a user-friendly, visual manner,
- configure and monitor network elements,
- detect failures of RCL components,
- create and configure the network services, and
- create subscriber entries.

For the QMTool it is thus necessary to intensively inter-work with the LDAP database, on the one hand, and with the RCL and network elements, on the other hand.

More details can be found in the chapters 3.4 and 5.5.

<u>Topics to be studied</u>

The following paragraphs discuss topics the project considers but only at *theoretical* level; their realisation might cause too much effort for the project. Analysing the requirements and proposing suitable approaches, the projects wants to build a flexible architecture that is *aware* of the below mentioned features in order to be open for later integration.

2.1.2.10 MPLS integration

The AQUILA approach and MPLS can coexist without necessarily having many interactions. Generally, MPLS is considered as a backbone network useful for both AQUILA QoS support and VPNs.

In the in this document proposed, basic "AQUILA over MPLS" scenario, the interaction between AQUILA and MPLS is in fact, an interaction between DiffServ and MPLS. Therefore it is mainly the task of the network operator – but not of the RCL – to define the mapping mechanisms between both worlds. Independent to this approach, VPNs may be used in parallel.

An advanced scenario may include two additional issues: the use of MPLS for AQUILA traffic engineering aspects as well as the support of QoS for VPNs.

More details can be found in the chapter 4.1.

2.1.2.11 Multicast

For IP multicast, two different cases has to be taken into account:

- Point-to-multipoint (p2m)
- Multipoint-to-multipoint (m2m)

With regards to the requirements, the project proposes detailed scenarios on how to extend the functionality of the RCL components, mainly the EAT and the ACA, for QoS multicast sessions.

For p2m, a stepwise approach is discussed for two kinds of multicast groups: *closed* ones and *open* ones. An m2m scenario increases the complexity even it can be split into several p2m scenarios. Especially for inter-domain multicast sessions an inter-working with the inter-domain routing and inter-domain resource reservation mechanisms is needed.

More details can be found in the chapter 4.3.

2.1.2.12 Security issues

There are four potential points, which might cause security problems:

- The signalling within the RCL as well as between the RCL and external entities. For the signalling within the RCL, which relies on CORBA, the use of CORBA security services is proposed. For the signalling between the RCL and external entities, other mechanisms such as SSH could be used.

- The access to the LDAP database. LDAP can be combined with other security mechanisms and protocols such as SASL and TLS.

- The DiffServ network. Two sub-aspects have to be considered: First, the border router between domains has to care of theft- and denial-of-service attacks. Second, the use of IPSec might be restricted due to the AQUILA approach for the classification of flows based on port numbers.

- The user authentication. User authentication is a task of the EAT, in combination with the EAT's web server, as well as the LDAP database. For the pure user *identification*, the EAT provides a login procedure which can be made secure by using well-known encryption protocols such as SSL. For the *authentication* however, special AAA protocols would have to be integrated. Also the LDAP database needs, due to its user data, completed by special security mechanisms.

More details can be found in the chapter 4.4.

2.2 Definitions

The following definitions extend the list of definitions in [D1201][1].

Advanced Reservation Mode. Reservation mode in which the technical oriented and the very detailed QoS reservation form that has to be filled. It is for a professional end-user that knows the meaning of each parameter of an AQUILA reservation request.

Application Profile. Unique description of one certain application containing in a hierarchical manner its network, technical, and session characteristics. Application Profiles aim at the storage of two levels of QoS abstraction: a network oriented, technical view, and a human oriented, more abstract (session) view. In that way, they allow a mapping between both levels.

Basic Internet Application. A usual Internet application such as Voice over IP, Video Conferencing, TV on Demand, FTP, Streaming, etc. A Basic Internet Application can be a standalone application or a web plug-in. It is often a legacy application, which is not QoS-aware.

BGRP Agent. A logical entity of the RCL. BGRP agents aggregate and communicate resource reservations between different network domains.

[1] Those terms that are underlined are either already defined in [D1201] or in this paper.

Complex Internet Service. A service offered by e.g. a content provider to a customer group in form of a web platform integrating, binding and presenting Basic Internet Applications to a combined, value added service. The use of web technologies (HTML, Plug-ins, Java, JSPs, etc.) allow the realisation of such a service, on the one hand, and offer access to QoS APIs as the one provided by the EAT, on the other hand.

Control Loop. Feedback from the network towards the RCL, in order to *close* the **Open Loop Control** of the first trial.. More precisely, the measured behaviour of the network elements is used to dynamically influence decisions taken by the RCL concerning the admission of flows (Measurement Based Admission Control) and the distribution of resources (re-provisioning).

Globally Well Known Service (GWKS). A certain traffic behaviour (in terms of traffic classes) that describes common, over domain boundaries accepted QoS objectives. More specifically, there are no fixed QoS parameters but optimisation targets such as “low delay”, as well as traffic ranges such as “maximum packet size”. In different domains, a GWKS might be implemented in different ways.

Measurement Based Admission Control (MBAC). Admission control that dynamically uses at network elements measured parameters to decide whether a flow can be granted the requested QoS or not. This is done in *addition* to the **State Based Admission Control (SBAC)** based on resource pools as it was realised for the first trial.

QMTool (QoS Management Tool). A software tool for the network operator, providing a GUI for the visual creation and modification of resource pools and network services, as well as the configuration and surveillance of network and RCL elements.

QoS Monitoring. The watching of QoS parameters in order to give a feedback to the customer who pays for the QoS, or to the application to adapt its reservation.

Protocol Gateway. In terms of AQUILA a **Proxy** that resides between the applications and the network in order to get information which might be meaningful to *complete* or *initiate* reservation requests for the applications. In this way, QoS can be provided also for (legacy) applications that either dynamically negotiate data port numbers (e.g. via H.323), or that use special (QoS) protocols (e.g. SIP, RSVP) to signal their QoS needs.
A Proxy can act as **Application Level Gateway** or **Network Level Gateway** depending at what protocol level it works.

Re-provisioning. The redistribution of resources by dynamically reconfiguring the resource pool tree.

Reservation Adaptation. Behaviour of an application itself, or of a QoS middleware to adapt the reservation requests to the real needs of the application, considering for example, the QoS monitoring results.

Reservation Group. An association of several single reservations that have a common context. Reservation groups can be established for bi-directional reservations (consisting of two uni-directional single reservations), for several reservations that belongs to the same application but different service components, or for multi-conference sessions, for example. Reservation groups are requested by a single message.

Resource Pool. The concept of distributing and sharing resources in a hierarchical tree structure. A Resource Pool consists of components that are either other (sub) Resource Pools or resource pool leaves. A Resource Pool is managed by an RCA, and visually created by using the QMTool.

Resource Pool Leaf. A component of a resource pool at the deepest level. In AQUILA, it is associated to an ACA.

Service Component. Part of an application profile, describing one element/medium of an application. For example, a multimedia conferencing application may have three service components: “video”, “audio”, and “data”. The purpose is to allow several, different reservations per application due to the different requirements of its media.

Usage Data. A collection of statistic data consisting such parameters as start and end time, duration, volume, etc. for a flow that belongs to a reservation.

Usual Reservation Mode. Reservation mode in which a non-professional end-user requests for a reservation by selecting a pre-defined, abstract QoS option. Such an option represents the technical behaviour of a QoS offer in a for usual end-users understandable manner.

Trusted Entity. A reliable network element (e.g. a router, an application, a server) inside the AQUILA domain. For such an entity, no authorisation and policing is necessary for (prioritised) signalling.

Part II: Architecture and studies

3 Overall structure and architecture

3.1 Resource control layer

3.1.1 Resource distribution

This section examines the creation of the resource pool hierarchy, its initial configuration as well as the resource distribution actually realised between the RPs entities. Based on the fact that the initially assigned resources to the RPLs/RPs do not reflect the real traffic load, an adjustment of those resources is needed. The basic principles for a dynamic resource re-distribution of resources assigned to RPs are examined.

3.1.1.1 Creation of Resource Pool Hierarchy

The distribution of resources is based on the realisation of the resource pools entities. The creation of RPs depends on the network topology, which in the access points of the backbone network usually follows a tree-like structure, which can be easily reflected to the hierarchical concept of the resource pools, the Resource Pool Tree (RPT).

3.1.1.2 Configuration of the Resource Pools

After creating the RPT, the distribution of resources is composed of two basic phases: initial configuration and re-distribution of resources. Therefore, initially resources are assigned to each RP/RPL according to traffic loads forecasts.

The term resource, as far as the resource distribution mechanism is concerned, compromises a single value parameter, namely bandwidth. Each RP/RPL is composed of a number of Resource Shares, each one managing the resources of a traffic class (TCL). Therefore, each RP/RPL has maximum 8 Resource Shares, per traffic class and direction (ingress and egress). Each Resource Share is configured with R_{max} and a R_{tot}, which define the maximum possible resources assigned to a RShare and the actually assigned resources respectively. Those values are retrieved from the LDAP server, where all the necessary values are stored.

The distribution of resources is performed in a top-down approach, where the root of the tree, which corresponds to the resources for the overall network, distributes the resources to its children, and the children distribute the resources to their corresponding children. This procedure continues till it reaches the leaves of the tree, the RPLs. A RP can distribute at most as much resources as it has received itself.

Another approach would be to assign zero initial resources to all the RShares of the RPs/RPLs, apart from the root of the RPT, which would have all the available network resources. The distribution of resources down to the tree hierarchy would be then based on the real traffic load, increasing though the interactions between the RPs.

Since, the initial distribution of resources is based on some static mechanism, those resources may be re-distributed based on the real traffic loads. The introduction of the resource pool algorithms encounters the dynamic distribution of resources between the RPs/RPLs focusing to a better network resources utilisation. A description of those algorithms can be found in deliverable [D1302]. Each RPL/RP needs some configuration parameters, including:

R	available resources of an RPL/RP
R_{max}	limit for resource assignment to an RPL/RP
R_{tot}	current resource assignment to an RPL/RP, with $R_{tot} \leq R_{max}$

In addition the following relations apply between a father RP and its children RPs or RPLs, where R_{res} corresponds to the reserved resources of an RP:

$$R^{f}_{\ res} = \Sigma R^{c}_{\ tot} \qquad (1)$$

$$R^{f}_{\ max} \geq R^{c}_{\ max} \qquad (2)$$

$$\Sigma R^{c}_{\ max} \geq R^{f}_{\ max} \qquad (3)$$

3.1.1.3 Re-distribution of Resources

The dynamic re-distribution of resources between the RPs/RPLs is based on some general rules and it is invoked when an ACA cannot accommodate a reservation request and its AC limit should be increased. Then, the ACA issues a request to its corresponding RPL. The interactions between the RPLs/RPs entities of the tree are depicted in Figure 3-1.

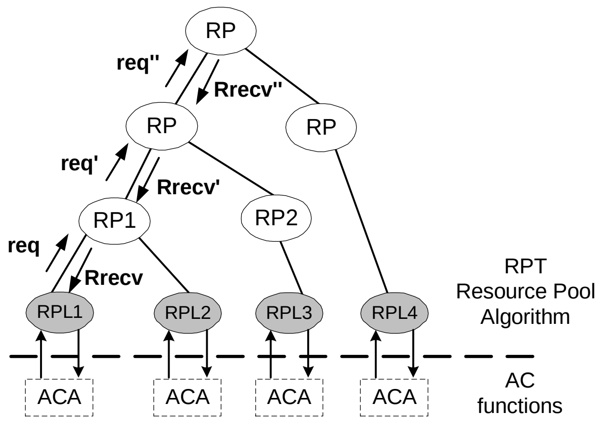

Figure 3-1: Interactions between RPs/RPLs

The RPL receives the request and calculates the new value of the AC limit based on its resources and the realised resource pool algorithm. If an RPL/RP does not have enough resources to accommodate the new request, then additional resources are requested from the above level, the father RP. Each RP/RPL runs the same algorithm, which is executed whenever resources should be re-distributed. The actual amount of returned resources from the father is based on its free resources as well as the implemented algorithm. The request for additional resources may be propagated till the root of the tree, as depicted in Figure 3-1.

A child may also return any unused resources to its father RP, if the reserved bandwidth is under a low watermark, *lw*. The amount of released resources also depends on the implemented resource pool algorithm.

3.1.1.4 Shifting Resources Between Traffic Classes

A crucial point for the second trial is the possibility of bandwidth shifting between traffic classes. After investigating the possibilities and the problems that the shifting of resources between Traffic Classes is introducing, that functionality would not be supported.

The basic problem encountered was the difficulty for changing dynamically in a router the weights of the WFQ in a short time scale. Apart from that, changing the weights of a single edge router would provoke the deployment of the new weights to the subsequent routers of the network.

An approach would be to change the WFQ weights only to the ERs, therefore re-distributing the resources between the ERs, while maintaining a static approach for the core network, which would be considered as over-provisioned. Even in that case changing the WFQ dynamically in a short time scale may not be possible.

Instead of shifting resources between the TCLs, the approach of control loops for re-provisioning the RPs in long time scale would be a solution. Changing the WFQ weights in a longer time scale may be possible and it is currently under study.

3.1.1.5 Resources: Inter-domain Issues

In case of the interconnection of different domains, each domain will manage its own resources and will distribute them to its RPs/RPLs. Each domain will also contain a number of Border Routers (BR), which will connect the domain to its neighbourhood domains. The BR will be configurable by the QMTool and will not be part of the RPT. For the assignment of resources to the BR, it is only taken into account the connection of the BR to its own domain. Moreover, they are actually controlled the resources, which the BR sends into or gets out of its own domain.

As far as the control of the inter-domain link is concerned, this is examined in section 3.2, where inter-domain resource allocation issues are encountered.

3.1.2 Network topology impact

3.1.2.1 Relevant Network Topology Issues

Even if trial scenarios will be limited, some aspects of real network topologies will be of interest for the 2nd trial. This chapter mentions some practical features of real large provider networks, while the next chapter tries to identify, which one can be realised during the 2nd trial with minimum effort.

3.1.2.1.1 Core Network Redundancy

One basic principle to gain high reliable networks is the doubling of the entire core network identically into an A and a B network. Compared to a hot standby system, here each core router location consists of identical entity pairs with equal configuration. They are permanently running. All links are doubled to the neighbour entities, e.g. 1A-2A and 1B-2B in Figure 3-2. Beside this the A and B entities of each location are connected directly to each other at least at the link speed connecting them with the neighbour entities as shown with 1A-1B in Figure 3-2.

To allow hot OAM work, e.g. release updates, the load of each entity must not exceed 50 % of the link capacity under normal circumstances. There is no congestion in the network if an entity or link fails.

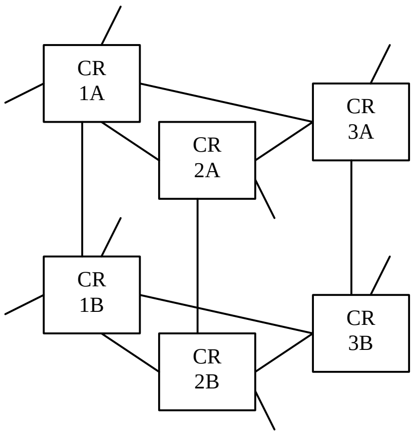

Figure 3-2: Parallel A and B network for high reliable networks.

3.1.2.1.2 Hierarchical Multiple Homing

For a high reliable network each edge router should have connections to at least two different core routers. To be aware of rapidly growing networks, a generic scalable architecture for the core is proposed. Starting from an inner core, which could be for instance a fully-meshed structure, like a triangle, additional core routers are dual homed to form an outer core as shown in Figure 3-3. To reduce the number of links instead of the triangle structure also loops can be designed for the outer core. This is especially preferred in areas with high local traffic. This architecture can be extended to several levels as needed.

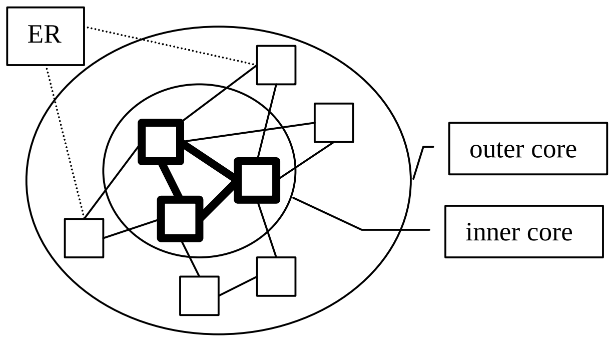

Figure 3-3: A hierarchical approach for dual homing of routers on several levels

The concept of dual homing can then also be applied to the connection of a host / customer premises equipment to more than one edge router, called dual hosting.

3.1.2.1.3 Access Networks

Real access network scenarios are very complex. Several functional components are working together to provide customers with a wide range of product portfolio. Such components are:

- Network Access Server (NAS)/Network Address Translation (NAT)
- Label Edge Router (LER)
- Broadband Remote Access Router (B-RAR)

Typically mixed IP/ATM infrastructure is performed. However some trends are typically:

- hose model accepted for resource calculation
- MPLS “grows” step by step to the edge (LER)
- specification of universal NAS/B-RAR/LER

3.1.2.2 Realistic AQUILA Scenarios

Due to equipment limitations trial scenarios are limited to basic functions, which prove flexibility and scalability of the modelling concept. There, basic scenarios will be combined to a basic AQUILA topology model that can be extend in several directions.

3.1.2.2.1 Dual Hosting of Hosts to Edge Routers

Usually a host or a LAN has a single connection point to the Internet world. Due to availability reasons, multiple networks and resource sharing dual hosting will be interesting. The selection of the used Edge Router (ER) will depend on several decision criteria.

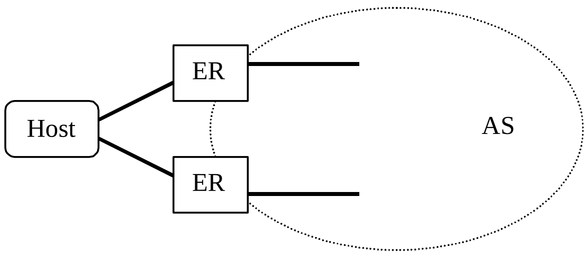

Figure 3-4: Dual hosting

Practical State of the Art: There are several reasons for dual hosting. In some cases Cisco's HSRP (Hot Standby Routing Protocol) is used and even though it would allow traffic sharing, it is practically not used. In general can be said that independent of used protocols, traffic sharing is not yet performed and the second connection is just a backup for the failure cases.

Impact on AQUILA: Independent of how the host selects the link for sending data, he should use this knowledge in order to use the proper interface source IP address for making reservations. If the host is connected to two domains, he should also use this knowledge in order to select the appropriate EAT. Reservations can then be carried out as usual.

3.1.2.2.2 Dual Homing of Edge Routers to Core Routers

An Edge Router (ER) can be homed by different Core Routers (CR). Dual homing has a deep impact on the Resource Pool calculation but increases the availability.

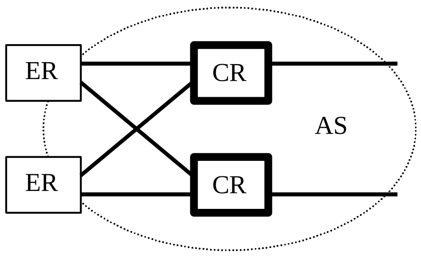

Figure 3-5: Dual homing of Edge Routers

Practical State of the Art: Different routing protocols are used and therefore also different strategies. Depending of the routing protocol traffic sharing, e.g. load balancing can be an issue. If one of the IGP protocols is used, traffic sharing is harder to realise. In case of BGP traffic sharing is rather easy to do and often also used. The traffic is shared statically between two links based on the IP-addresses (sub-networks).

Impact on AQUILA: If an edge device is connected to multiple core routers, whose resources are not managed by the same higher level pool, a more complicated situation arises. A possible solution is to allow a resource pool leaf to have multiple parents. Even if the dual homing is performed on a static basis, it is still unknown, how much traffic uses which route. Measurements could provide some percentage values on how much traffic of the leaf is sent to which parent. Requests to the parents are then always made using these percentages. E.g. if measurements find out, that 70% of the traffic is going over link 1 and 30% over link 2, then the resource pool leaf will always request resources in this relation from the two parents, if it runs out of resources. Alternatively, as the decision which route to take is mainly based on the destination address, the decision rules (excerpt from the routing tables) could also implemented in the RCL components.

For implementation, the project assumes that the network operator or some other mechanism controls the amount of traffic taking each route. A statically configured percentage for each link is used to split the reservations among the multiple parents.

3.1.2.2.3 Dual Peering Points

Peering points between different networks (autonomous systems) are equipped with Border Routers (BR). Usually more than one peering point between two selected providers will exist. This influences the information exchange about available resources. A summary contract has to be fulfilled but resources have to be assigned per peering point bilaterally.

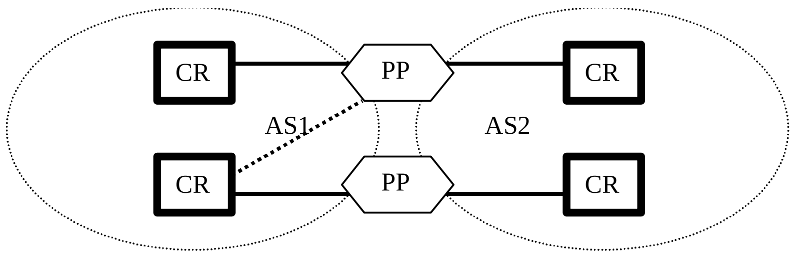

Figure 3-6: Dual peering points between autonomous systems

For integrity reasons each peering partner will run its own BR entity. This means a peering point consist of two corresponding routers. Dual homing of BRs to CRs can be taken into consideration.

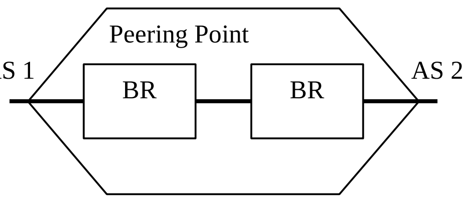

Figure 3-7: A Peering Point consists of two Boarder Routers

Practical State of the Art: Usually peering points between peering partners (ISPs) are limited to reduce contractual issues. Nevertheless multiple peering points omit single points of failure and help not to concentrate traffic at single points, which usually requires huge routing engines. Using the network of a peering partner, additional loops can be created to complete local topology structures. In case of transatlantic partners this becomes more and more practice as bandwidth is available and strategic partnerships are formed.

Impact on AQUILA: BGP can always find out, which BR is actually used. So dual peering adds no complexity to inter-domain resource control. Typically, both BR advertise the reachability of the same AS (and the addresses behind). Local policies prefer the one or the other peering point, depending on constraints the network operator defines. These policies are used to configure BGP. So BGP can determine the correct BR also for dual peering.

3.1.2.3 AQUILA Basic Topology Model

The AQUILA trial architecture tries with a minimum of equipment to proof typical topology situations with multiple instances. Dual hosting, dual homing and dual peering are combined in a single basic topology model.

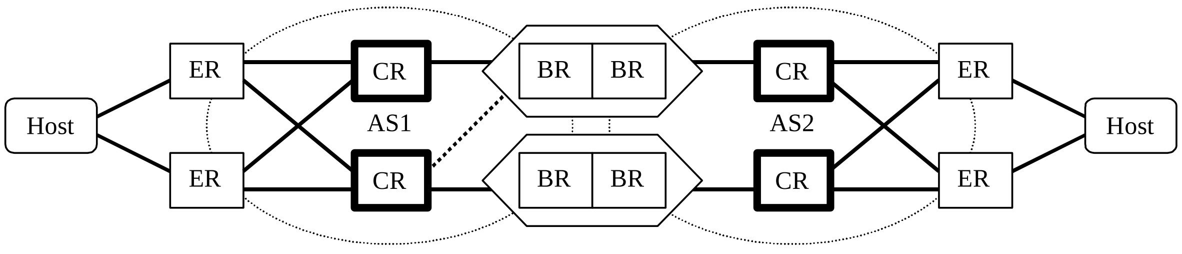

Figure 3-8: AQUILA basic topology model

3.1.3 Control loops

3.1.3.1 Introduction

Within the AQUILA architecture, the Resource Control Layer (RCL) is responsible for two major tasks: (1) the resource monitoring and distribution and (2) the admission control. The resource distribution task is further split into initial resource provisioning and resource pool management (dynamic adjustment of resource distribution in the network). The admission control is responsible for controlling the user access to the network in order to guarantee QoS service objectives. Current implementation of RCL is generally based on open loop control i.e. the resource provisioning or admission control is performed without taking into account any feedback information from the network (link utilisation etc.). One exception to this is Resource Pool mechanism that implements a sort of control loop by dynamic shifting resources between ACA (in dependence of the demands at certain points in the network). Design of more control loops for enhancing RCL layer performance is one of the goals of second trial.

RCL architecture designed for the first trial generally assumes static resource distribution both between traffic classes and inside traffic class (with some exceptions related with resource pool management). The network resources of given traffic class are distributed between resource pool trees by off-line calculation procedure (initial resource provisioning). The resource pool concept introduces some degree of dynamic in resource distribution. It allows dynamic resource redistribution between ACA agents that belong to the same resource pool tree. However this resource redistribution is limited only to some parts of the network, where depending on network topology, the resource pool hierarchy could be defined. In the second trial dynamic reconfiguration of core network resource distribution, based on some feedback from the network, is planned to be implemented.

The admission control algorithms implemented in the RCL layer (by ACA agents) accept or reject flows on the basis of information provided by the End-user-Application-Toolkit (EAT) (for legacy applications) or by the application itself (for QoS-aware applications). The ACA agent uses this information to decide whether enough resources for the connection request are available in the network (in hierarchical Resource Pools structure). In AQUILA the stochastic nature of the packet traffic is described in the form of deterministic parameters that corresponds to a token bucket algorithm. Moreover the admission decision is based on the parameters specified before the any real user traffic is send to the network. The AC methods based on token bucket description are usually very conservative since they consider the worst-case traffic pattern (in practice one can observed that the real traffic is quite far from that assumed for AC). As a result, these methods do not guarantee effective network resource utilisation. The performance of ACA (the resulting network utilisation) can be improved by using feedback from the network.

This chapter specifies the system architecture for implementing measurement procedures for supporting RCL tasks mainly the admission control and initial resource provisioning. No specific algorithms for these tasks will be presented below instead general measurement architecture that can support these algorithms will be described (the actual algorithms for realising control loops in RCL will be proposed by WP1.3).

3.1.3.2 Closed loop versus open loop control

Current RCL layer uses open loop resource control algorithms for resource provisioning, and the admission control. This control scheme is depicted on Figure 3-9. In the considered case the resource distribution or resource assignments to the user requests is done only on the basis of a priory specified parameters (e.g. user declarations in case of admission control or assumed traffic demand in case of provisioning algorithms). In the closed loop control these tasks are perform not only on the basis of the assumed or declared parameters but also on the basis of the parameters obtained from the network by some measurement procedure. The feedback from the network forms control loop.

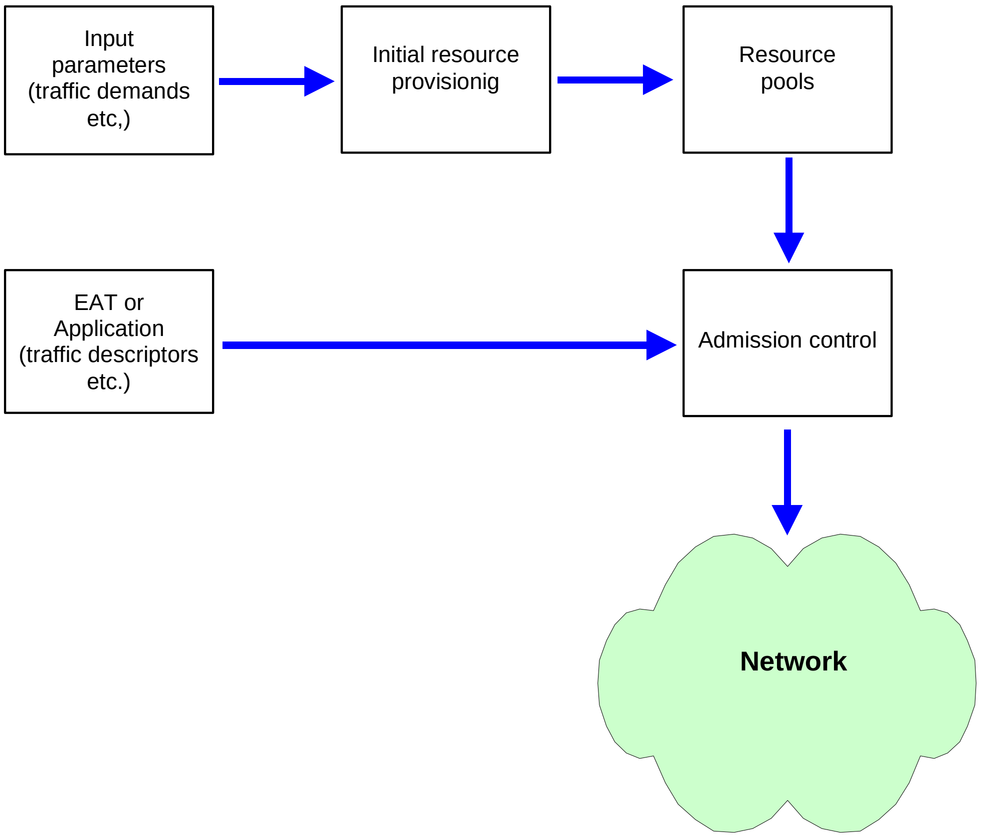
Input parameters (traffic demands etc,)
Initial resource provisionig
Resource pools
EAT or Application (traffic descriptors etc.)
Admission control
Network

Figure 3-9: Resource control in the first trial (open loop).

The closed control loop scheme is depicted on Figure 3-10. In case of AQUILA network this type of control can be used in admission control or resource provisioning algorithms. In order to realise control loops the supporting measurement architecture has to be defined.

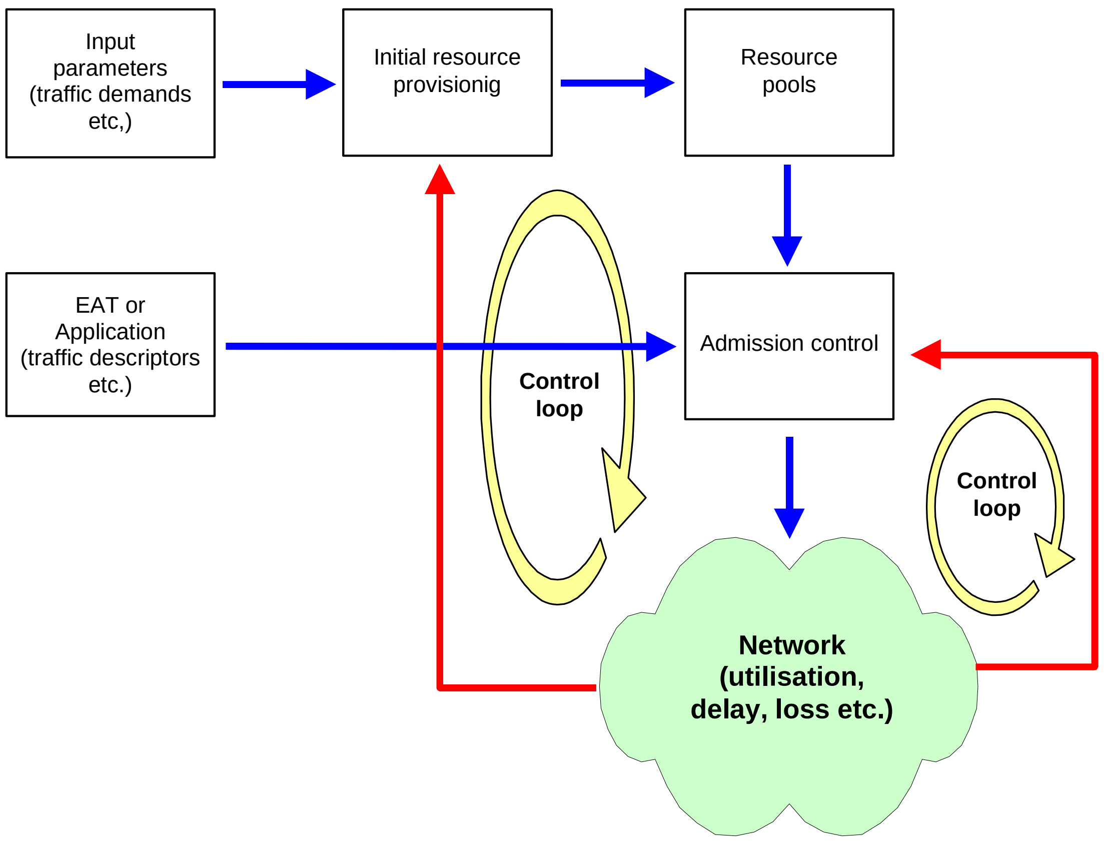
Input parameters (traffic demands etc,)
Initial resource provisionig
Resource pools
EAT or Application (traffic descriptors etc.)
Admission control
Control loop
Control loop
Network (utilisation, delay, loss etc.)

Figure 3-10: Resource control in the second trial (closed loop).

3.1.3.3 Admission control loop

Most of the measurement-base admission control algorithms require the measurement of the offered traffic rate. In case of AQUILA architecture this parameter has to be measured on per traffic class and per ED link basis and per ingress and egress direction (in case of p2p reservations scheme). For this purpose the passive monitoring architecture seems to be the most convenient solution.

The proposed admission control loop architecture is depicted on Figure 3-11. Each ACA agent is associated with one MBAC component. The following design assumes that MBAC is an integral part of the ACA agent. The MBAC consists of one MbacMonitor, which is responsible for retrieving data from the router and one or more MbacProcessors, which process the data obtained from MbacMonitor and implement the AC algorithms. The MbacMonitor regularly polls the router statistics for the offered traffic (number of bytes) on the set of links between its Edge Device and first Core Routers in each traffic class. On the basis of these parameters the each MbacProcessor agent can calculate the parameters required by the given admission algorithm e.g. temporary mean traffic load in each class (the mean load in this measurement interval) etc.

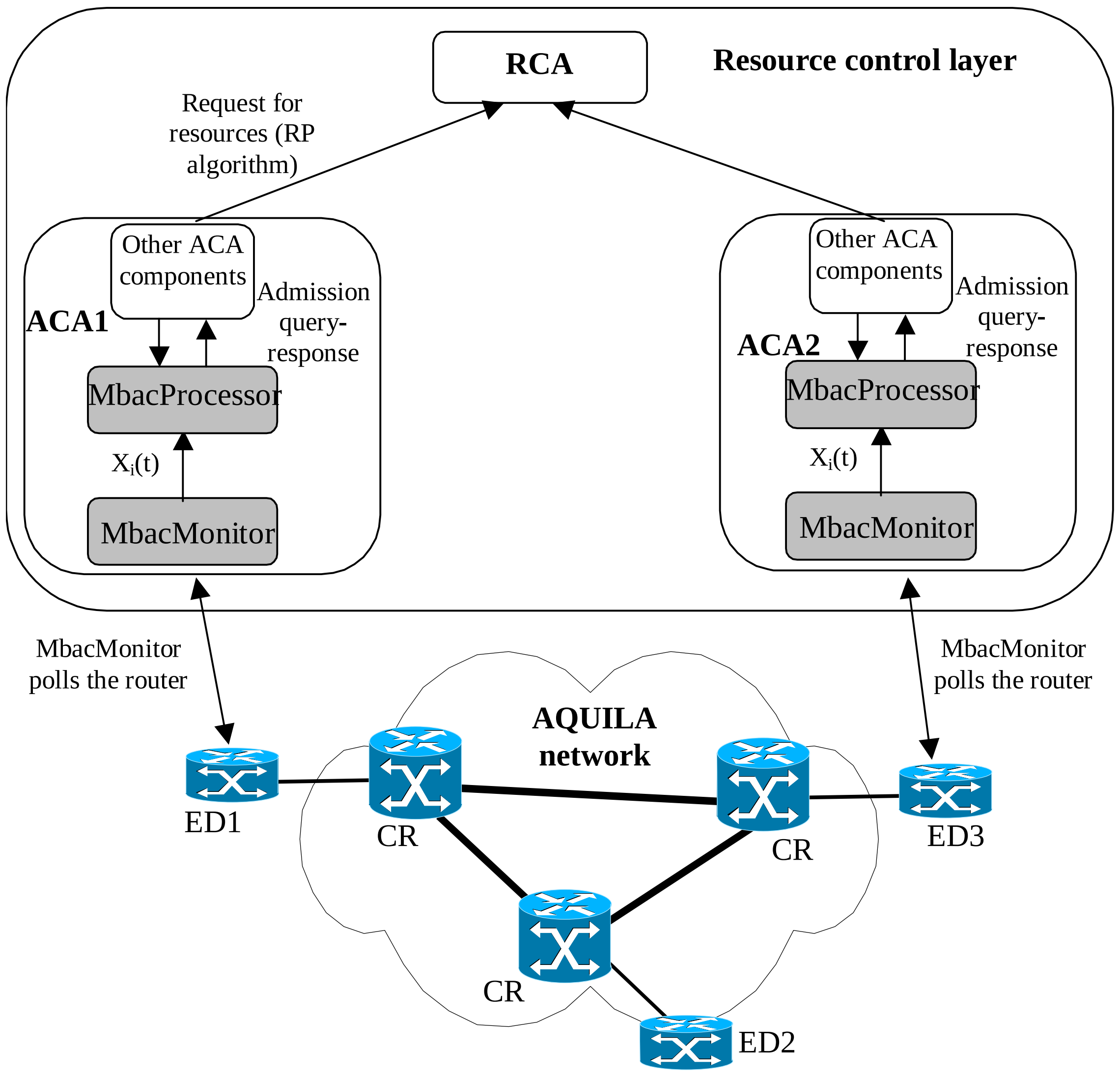

Figure 3-11. Measurement architecture for admission control loops

The **MbacMonitor** measures the traffic rate offered to the link controlled by given ACA agent (associated with AC limit value).

The selection of interfaces (and physical routers) for obtaining the statistics by the MbacMonitor may depend on whether the MbacMonitor have to measure the ingress or ingress and egress traffic. The MbacMonitor separately measures traffic for ingress and egress direction. The egress traffic is measured if the traffic class uses p2p reservation. Currently this scheme is used in case of TCL1, 2 and 3 class.

An example of the ED, which has to be monitored on the ingress and egress side, is presented in Figure 3-12.

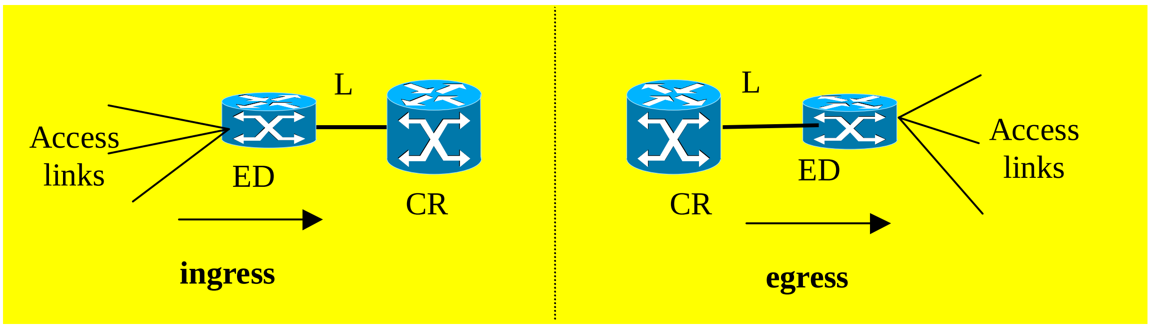

Figure 3-12. Links monitored by the MbacMonitor on the ingress and egress side of the ED

Assume that the ACA controls link L on the ingress as well as egress side (for TCL1, 2 and 3 traffic classes). On the ingress side the MbacMonitor should monitor the traffic send from ED to the CR. The MbacMonitor reads the statistics from outbound interface of the link L from ED. The data has to be obtained from the scheduler statistics on the specified interface. In this mode the transmitted and dropped packets (bytes) should be taken into account. Remark that the concrete method of obtaining the traffic statistics can depend on the router type (e.g. in Cisco it may not be possible to read statistics from the CBWFQ scheduler, so it will be necessary to set “artificial” policers on the ED’s outbound links and read data from the “show int rate” command).

For the egress direction, the MbacMonitor should monitor the traffic send across the link L from CR to ED. The statistics are read from inbound interface of the ED. This may require additional configuration of the inbound interface (setting of “artificial” access-lists and rate-limits).

In general, each MbacMonitor should keep the list of routers and interfaces (together with the information on the type of monitored data), form which it should read the statistics in order to obtain offered load $X_i(t)$ for a given link.

Every specified measurements period the MbacMonitor sends new rates $X_i(t)$ for each monitored link, each TCL (i=1...4) and ingress and egress direction (where appropriate) to its MbacProcessors and resets the router statistics. The resetting of router statistics can be avoided if the MbacMonitor would know the maximum value of the statistic counter and detect the counter overrun. The value of the measurement period duration should be a configurable parameter (possibly initially configured with the use of the QMTool). The length of the measurement duration can be the same for each TCL class.

The **MbacProcessor** makes the actual admission decision by implementing a respective admission control algorithm. Separate MbacProcessor exists for each controlled link, for each traffic class and ingress and egress direction. The operator should have the possibility to specify, whether the admission decisions in a given ED and in a given traffic class should be performed using MBAC (then, the AC queries are directed to the MbacProcessor) or using the declaration based method (then, the AC queries can be handled in a similar way as in the first trial). The tuneable parameters of algorithms implemented in the MbacProcessor should be initially configured with the use of the QMTool, separately for each TCL class.

MbacProcessor can be requested by other ACA component to make an admission decision for a new flow. Therefore, all MbacProcessors have to implement common interface, with method AC(), which returns a simple yes/no answer to the admission request. The type of interface (Java or CORBA) the MbacProcessors have to implement depends whether other ACA components have to communicate with the MbacProcessors via network or not.

3.1.4 Reservation groups

The establishment of some services needs a multiple reservation in the network. To support this requirement an extended concept of reservation is included in the AQUILA structure [D1201]. A good example is a multi-conference where a collection of reservation requests is necessary. The idea of reservation groups is to associate all these requests and send a unique request with all the reservation information. This feature may also be necessary in order to support TCP traffic because the QoS for ACK packages in the backward direction does influence the forward data stream.

The request association is done at the Requester EAT where all the information about the individual reservations is known. This reservation group request is sent to the Manager ACA, which extracts the information needed for requesting the individual reservations. With this structure the impact over the global architecture is minimal. The main changes are focused on the EAT. When the service finishes the Requester EAT release all the connections corresponding to the reservation group.

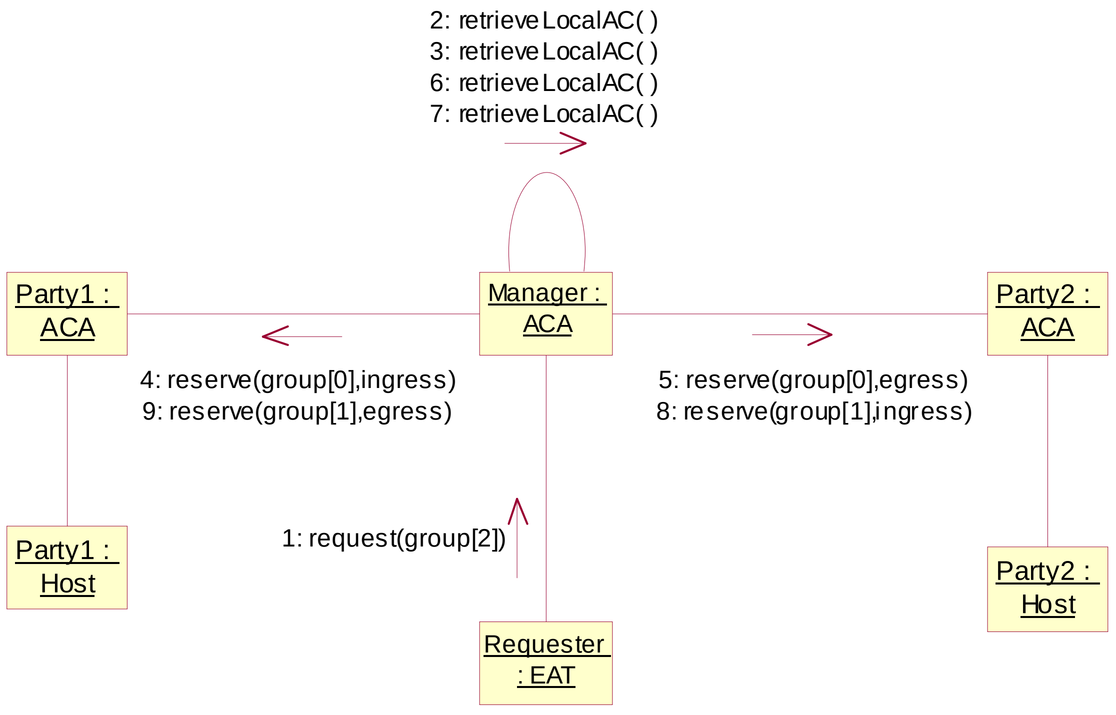

Figure 3-13: Bi-directional reservation request using Reservation Groups

In the above figure an example is shown for a bi-directional reservation. The Requester EAT creates a list of connections ***group***[]. Every element of ***group***[] contains the information needed for making a reservation. The steps sequence of the reservation is:

- The Requester EAT sends a request to the Manager ACA with a list of 2 reservations (*group[2]*) from Party1 to Party2 and vice versa. *(1)*
- The Manager ACA determines the Ingress and Egress ACA for the first direction. *(2,3)*
- The Manager ACA reserves the first direction from Party1 to Party2 that corresponds to the information included in the first element of the reservation list (*group[0]*). The ingress node corresponds to Party1 and the egress node to Party2.*(4,5)*
- The Manager ACA determines the Ingress and Egress ACA for the second direction.*(6,7)*
- The Manager ACA reserves the second direction from Party2 to Party1 that corresponds to the second element of the reservation list (*group[1]*). The ingress node corresponds now to Party2 and the egress node to Party1. *(8,9)*

The steps (2,3) and (6,7) could be unified for this example of reservation group since ingress and egress ACA are just changing the roles but it can not be extended to other reservation groups with different characteristics.

The multiple reservation request brings up a new topic about what to do when not all the reservations are successful. The different approximations based on different services could take us to very complex algorithms, which are not the objective of this project. A very elemental algorithm is to include a parameter "***complete_group_needed***" in the reservation group request which specify if all the individual reservations are indispensable for the global reservation. When this parameter "***complete_group_needed***" is activated, if one of the reservations fails, all the established connections will be released and a failure message will be sent to the Requester EAT. When "***complete_group_needed***" is not activated the failure information would be transmitted to the Requester EAT, which could ask the user for action or answer according to pre-established schemas depending on the service.

In order to provide a flexible structure it must be possible to add or to release reservations to the reservation group. The Requester EAT will create a "***join***" message with all the information needed by the Manager ACA for adding a new reservation to the existing reservation group. In the same way a "***release***" message is added in order to cancel all the reservations which are not needed anymore and to let free the unused resources but without releasing the complete reservation group.

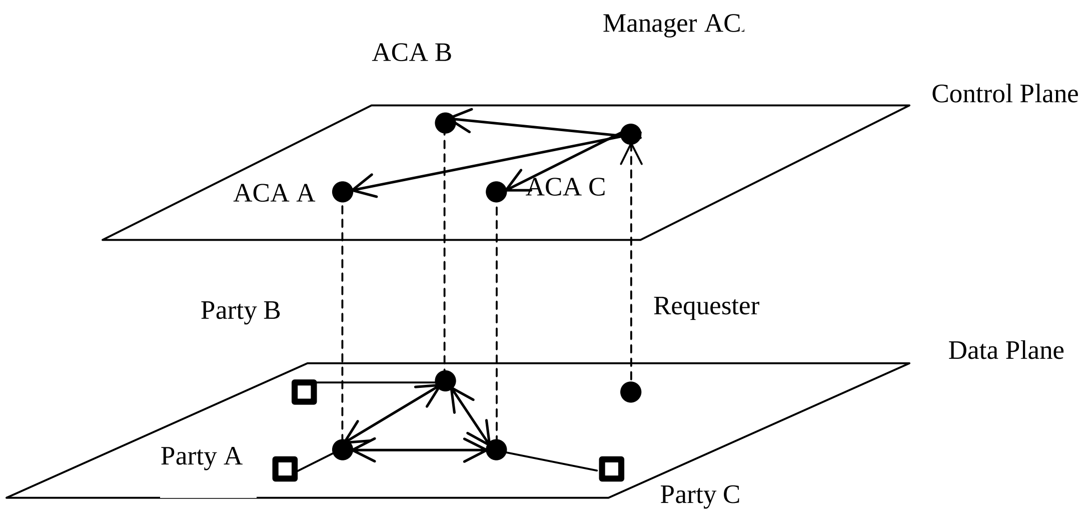

Figure 3-14: Example of reservation groups in a multi-conference scenario

Due to the presented schema a service using reservation groups is thought as a group of individual connections, and is far away from a centralised idea, where all the users are just connected to a distributor. The Requester EAT and the Manager ACA are just administrating the reservations, they are just working within the control plane (RCL). In the same way, when a new reservation has to be added to the reservation group the Requester EAT is the only one, which can generate the "***join***" message. Returning to the multi-conference example, the figure below shows how a new user (Party D) joins the reservation group. The join message can not be send by the new user. The message has to be sent by the Requester EAT which has all the information about the reservation group and is the one which will be charged by the reservation. The Manager ACA has to send reservation requests to all the ACA implied in the multi-conference in order to connect the bi-directional communication Party D with all other multi-conference participants.

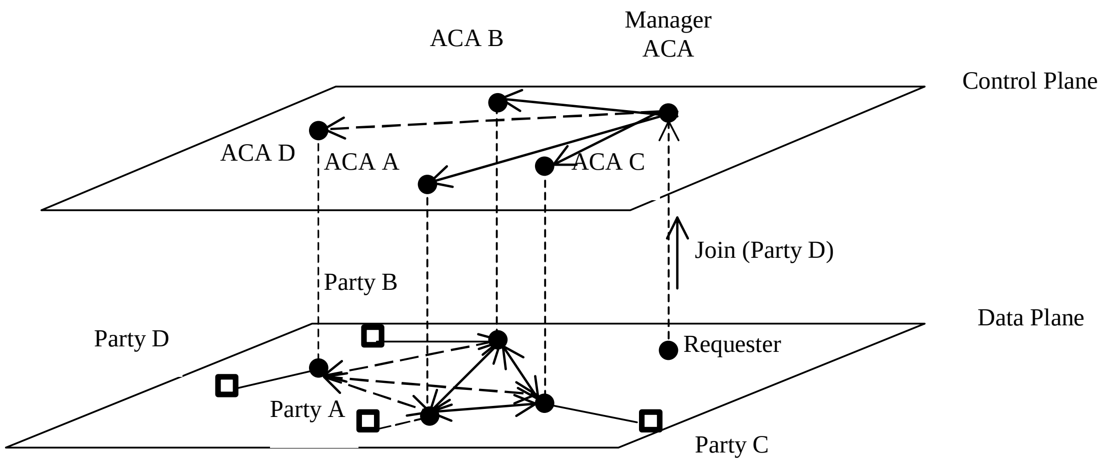

Figure 3-15: Example of join message in a multi-conference scenario.

3.1.5 Prioritised signalling and control traffic

The AQUILA architecture is constructed as an overlay to a DiffServ aware network. The resource control layer -RCL- is a logical layer with connections to the DiffServ net in order to control and to communicate with the QoS environment (DiffServ aware ED/CR/BR). For this inter-working with the DiffServ network, signalling traffic or control messages of the RCL traverses the DiffServ domain (nameable AQUILA domain) up to the controlled entities and vice versa.

It is assumed that this traffic is carried about the same physical links as the data load. In this case the signalling traffic rivals with the prioritised or best effort traffic for the available bandwidth and in an DiffServ based IP network only a prioritisation mechanism has to be developed in order to guarantee that specific traffic flows aren't blocked by the other data flows.

The Figure 3-16 shows the RCL as a logical overlay net with some physical connections to a DiffServ aware network. The lines symbolise a link between two entities.

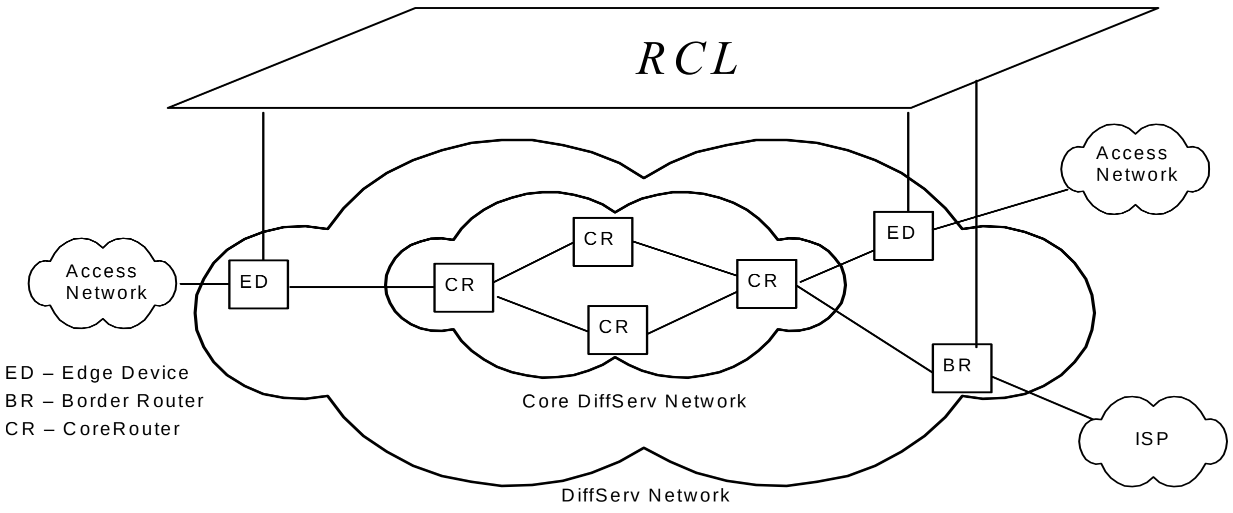

Figure 3-16: AQUILA Resource Control Layer

Due to the AQUILA overlay network, signalling traffic occurs <u>within the RCL</u>, <u>between the user and the RCL</u>, <u>between the RCL and the DiffServ network</u> and <u>between different domains</u>.

Be aware that the RCL is a logical layer realised by software components e.g. Figure 3-17 that are hosted on different machines which can be located theoretically anywhere. The actual localisation of the components affects some of the following considerations. However we have carefully tried to cover most of the possible scenarios.

3.1.5.1 Signalling traffic within the AQUILA Domain

3.1.5.1.1 Control messages within the RCL

First the signalling traffic within the resource control layer shall be investigated. As mentioned above the RCL is a logical layer. It is hierarchically structured and each component owns a CORBA interface in order to communicate with another RCL components. So a middle-ware layer transports the control information between the RCL elements. Figure 3-17 shows the logical bi-directional signalling paths within the RCL.

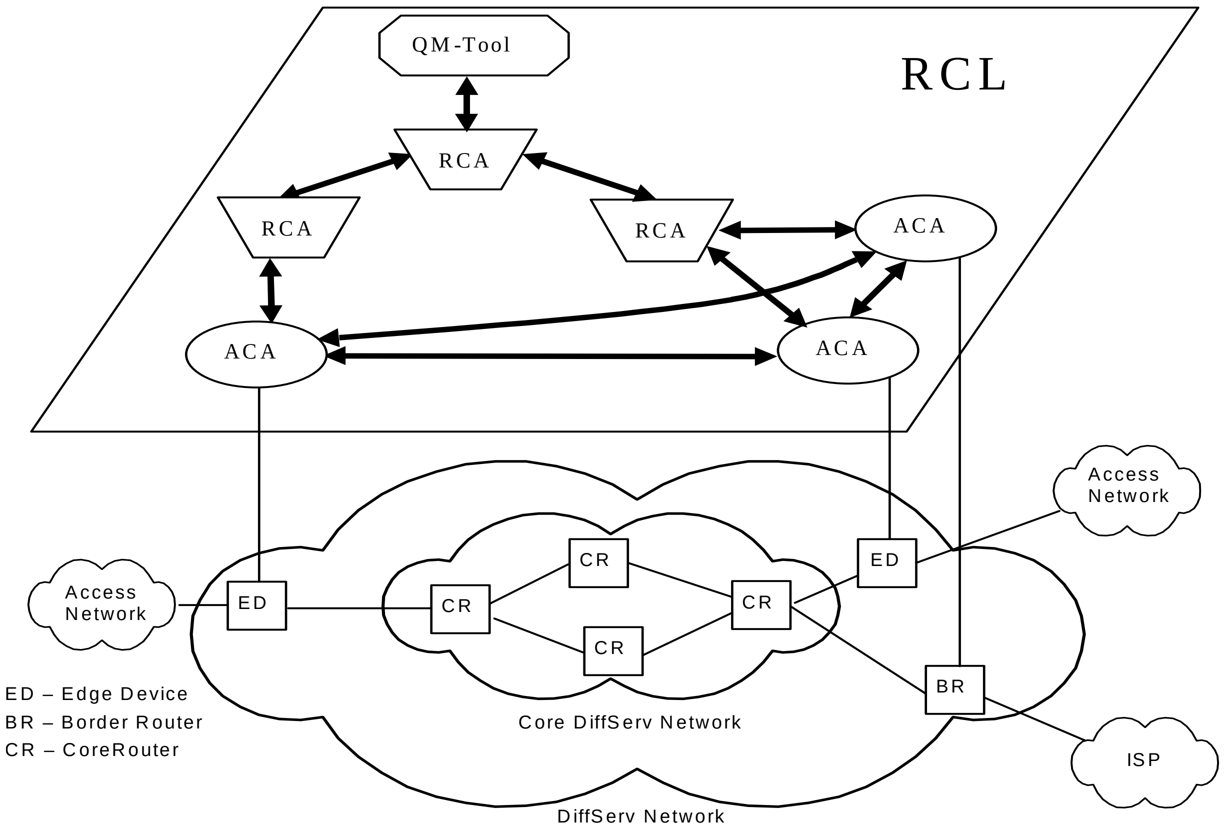

Figure 3-17: Signalling flows within the RCL

In order to separate the signalling traffic from DiffServ data load it would be possible to host the software components of the RCL on entities that are strictly separated from the DiffServ network. I.e. by using a separate signalling network with some connection points into the DiffServ net e.g. between ACAs and EDs or BRs.

But the AQUILA architecture does not introduce such a separated network for the Resource Control Layer and therefore no protected network for the signalling traffic exists.

This implies that the ACAs are located next to this EDs and the signalling traffic between the ACAs and from the ACAs to the RCA traverses the DiffServ network and has to be prioritised.

The RCA itself can be hierarchically structured and each part of the RCA can be distributed within the AQUILA network, always under the premise nearest to the communication partners within the RCL, i.e. nearest to the ACAs.

And also the communication traffic between these “RCA-sub-layers” has to be based on prioritised signalling traffic.

3.1.5.1.2 Control messages from the RCL into the network

For the inter-operation between the RCL and the network entities edge device adapters, a subsystem of the, ACA is designed.

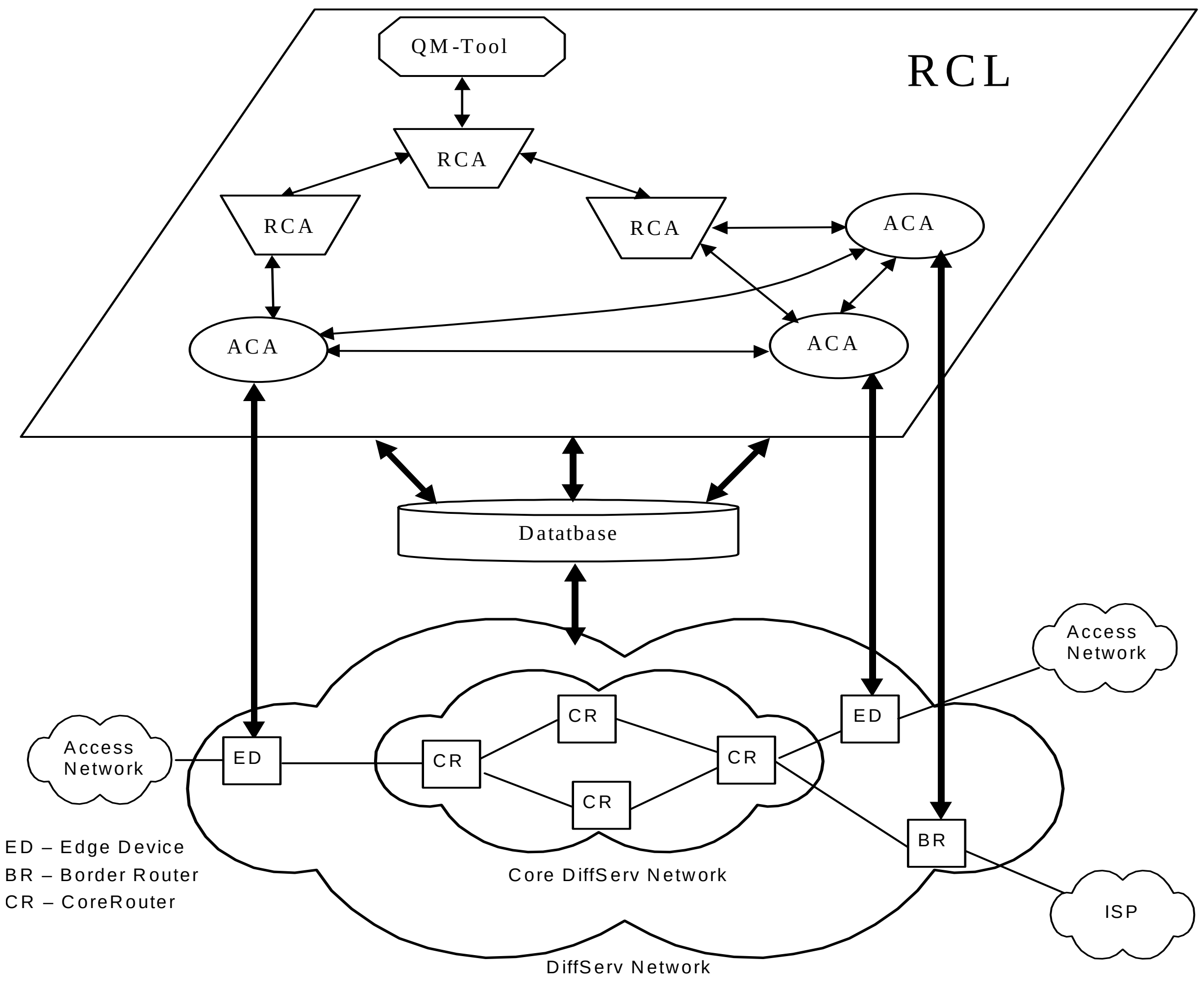

Figure 3-18: Signalling traffic from the RCL into the DiffServ domain

Between the network elements and the ACAs no middleware like CORBA is used, but this communication streams are based on protocols like TELNET or SNMP in order to control and configure real edge routers available on the market.

However this signalling traffic between these entities has to be prioritised or a dedicated line like a direct control connection has to be installed.

The other type of communication traffic from the RCL into the DiffServ network is the inter-working with management components of the DiffServ net like databases, accounting post-processing entities, policy server etc. The configuration properties of the AQUILA RCL are stored on LDAP database and dynamically used from all the RCL components. This signalling traffic has to be prioritised on the RCL side and also on the installed legacy equipment.

3.1.5.2 Signalling traffic into and out of the AQUILA domain

a) Signalling traffic from the user to the RCL

This chapter covers the signalling traffic that the user's inter-working with the RCL layer spawns.

If the user sets up a communication to the EAT (End-user Application Toolkit), signalling traffic flows from the user to the EAT. Each EAT is located within the DiffServ domain as each EAT is logically bound to one ACA and contains central services for all users that are physically connected to the ED controlled by this ACA. In this scenario (Figure 3-19) signal-ling traffic to the RCL crosses the DiffServ boundary at an ED and has to be recognised there and for prioritised handling.

There are three different kinds of signalling streams from the user's PC up to the EAT that enters the DiffServ domain

- HTTP requests to a WEB server to access the AQUILA QoS portal.
- Communication requests of the EAT's proxies or sniffers to the EAT.
- QoS requests from the QoS aware applications directly to the EAT API.

These traffic and signalling flows have to be recognised, policed and marked at the ingress ED of the DiffServ domain.

In order to recognise these traffic streams the destination IP address is the only applicable identification criteria. The destination IP address has to be the IP host address of the EAT, which is logically bound to the ingress ED or ACA, or the IP address of central entities like the AQUILA QoS WEB portal server.

But nevertheless only an expected amount of signalling traffic to the EAT or to the QoS Web server might be accepted and therefore policing rules are mandatory. The other identification points like Destination Port, Source IP address, Source IP port or a pre-marked DCSP give no further information. Only the destination IP address is helpful.

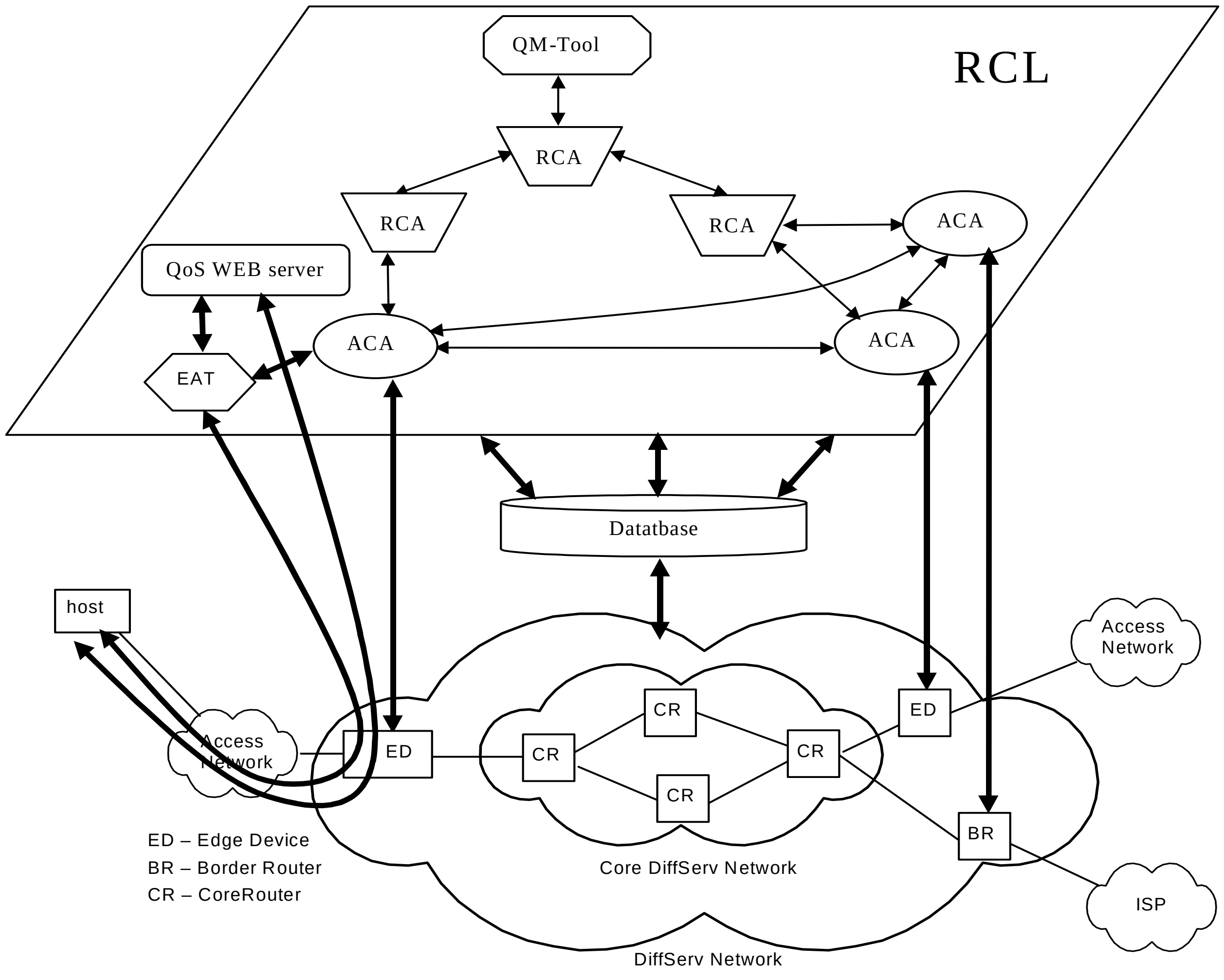

Figure 3-19: Signalling flows from access net into an AQUILA domain

b) Signalling traffic into adjacent domains

Two different kinds of signalling traffic has to be distinguished

- the hop by hop signalling traffic
- the routed signalling traffic

An example for the hop by hop signalling traffic is the BGRP traffic and this traffic has to be handled in a prioritised manner like other hop by hop protocols e.g. BGP or RSVP.

The routed signalling traffic occurs if a AQUILA RCL signals e.g. in an other not adjacent AQUILA domain in order to transfer control information into the destination domain for a AQUILA intra-domain reservation.

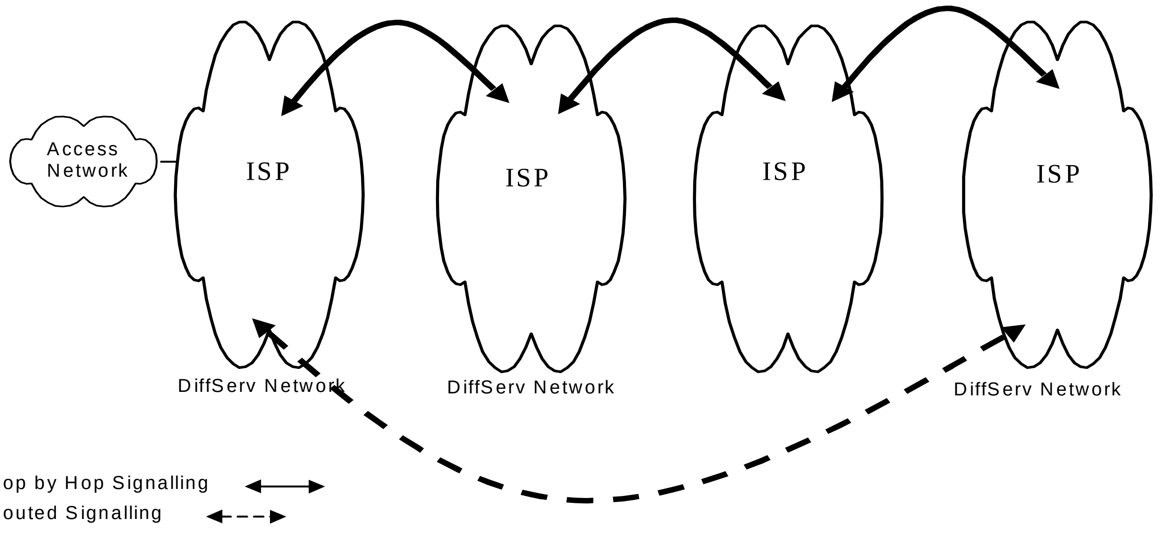

Figure 3-20: Signalling flows between domains

With the focus on this routed signalling traffic a “global well known signalling service” is needed in order to install prioritised signalling connection between distributed but logical connected control layers. Therefore the routed signalling traffic into the adjacent domains has to be marked with an especial DSCP, that signs this prioritised traffic.

The task of AQUILA domain is to observe the fixed agreements and to protect these. The adjacent domains have to recognise and transfer the AQUILA routed signalling traffic as arranged.

3.1.5.3 Marking of the signalling traffic

As explained in the chapters above, signalling traffic arises by the AQUILA RCL

- within the RCL between each component
- initiated at the user to communicate with the RCL and also backward
- between the RCL and the entities of the DiffServ network and also
- inter-domain signalling traverses the net

In IP networks the data load and signalling load use the same links and therefore the signalling traffic has to be prioritised at the ingress point of the AQUILA network or on the host machines on that the control traffic is generated.

One premise is that all the entities within the DiffServ network are trusted and the traffic type generated at these machines has to conform to the requisites of the DiffServ AQUILA network (like valid DSCPs, expected traffic load etc.). Then these actions has to be done

- the signalling traffic inside the RCL has to be marked at each host of a RCL component,
- the signalling traffic from the user to the EAT has to be recognised, policed and marked at the ingress ED including the user's http requests to the QoS WEB server
- the signalling traffic to the ED and backward to the ACA must be marked or the ACA is directly connected to the ED
- the trusted signalling traffic into the adjacent net has to be marked with a DCSP according to the SLA with the neighbour domain operator

Different solutions for the handling of the marked traffic exist

1. mark the signalling traffic with the same DSCP used for one of the existing traffic classes and mix the signalling traffic with that prioritised type of traffic
2. set-up a new traffic class for the signalling traffic with an own scheduling queue
3. set-up a new DSCP for the signalling traffic but mix this traffic into an other scheduling queue
4. use the same DSCP as the routers use for the router to router signalling protocols
5. or fully mash the RCL components with MPLS paths

The signalling traffic is characterised by the requirements of low bit error rate, low delay, high priority, bursty traffic and possible strong variation of the packet size.

However the provisioning has to consider this signalling traffic in any way and a traffic engineering model has to be set-up for the AQUILA domain but further studies of work-package 1.3 will cover this topic.

First of all it is necessary to recognise the AQUILA signalling traffic inside the AQUILA domain in order to be able to modify dynamically the policing mechanisms set in the domain regarding this type of traffic. It is also important for the traffic engineering to get information about the AQUILA signalling traffic load and distribution within the domain. The AQUILA measurement tools e.g. have to get the possibility to watch this traffic type. Therefore on the IP layer a typical ToS marker has to be set and therefore a new DSCP should be created.

But if the signalling traffic is mixable with other traffic streams or better separated or the handling of the classifier, marker, policer has to be discussed in WP1.3.

3.1.6 Logging of usage data

The AQUILA architecture provides packet-based communication services for the users. The users have SLAs with their ISP and depending on these SLAs the users have to pay for the demanded network services and the offered Quality of Service (QoS) by the ISP. Therefore the AQUILA architecture has to make available an infrastructure for the charging and accounting.

3.1.6.1 Billing, charging and accounting

Current pricing and charging methods for the Internet are not based on actual usage of this service, which leads to unfairness and delivers no feedback through financial incentives to network provider [FANKH]. With the Next Generation Internet not only technical services like bandwidth reservation will be introduced.

Figure 3-21 shoes the relationship between different tasks of the charging, accounting and billing process [ETSI734]. The AQUILA architecture just represents the area of technical relevance. The areas of administrative, strategic and market relevance are out of scope for the AQUILA Project.

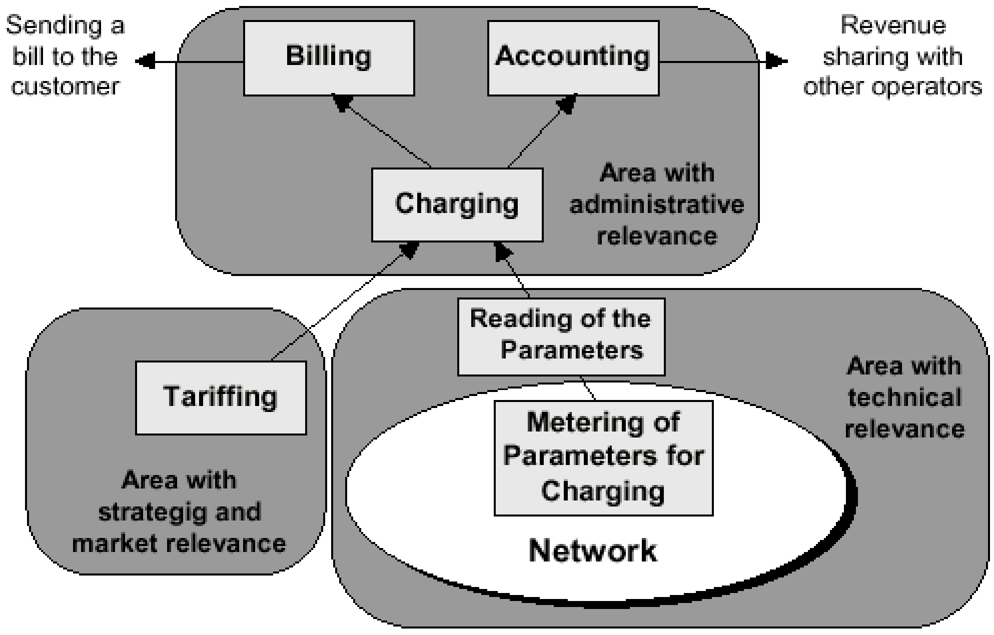
Sending a bill to the customer
Billing
Accounting
Revenue sharing with other operators
Charging
Area with administrative relevance
Reading of the Parameters
Tariffing
Area with technical relevance
Metering of Parameters for Charging
Network
Area with strategig and market relevance

Figure 3-21: Relationship between the different tasks in the charging/accounting/billing process

Charging is based on metering the usage of network resources. Monitoring describes the lowest layer of the infrastructure required where; information is gathered about what happens in a network. In the packet-based network architecture monitoring is based on packets or datagrams that travel independently through the network. These packets contain information about the amount of data transferred and allow counting the number of bytes sent.

Monitoring packets itself is not difficult [FANKH]. It rather depends on the processing that is done on these packets and, therefore, sampling the packets may be an alternative to strictly counting packets. These mechanisms are quite common and are used to implement volume based charging. Other models concentrate on priority or congestion resolution, which requires the monitoring layer to obtain information about many packets on certain conditions. The information required by charging and accounting and these conditions must be freely configurable.

In the IETF accounting is part of the AAA working group. In different drafts and RFCs accounting basics [MILLS] and the use of RADIUS or DIAMETER [BROW] for accounting are suggested. These workings adopted the accounting framework and terminology used by OSI (ISO 7498-4 OSI Reference Model Part 4: Management Framework). This framework defines a generalised accounting management activity that includes calculations, usage reporting to users and providers and enforcing various limits on the use of resources. The OSI accounting model defines three basic entities:

1) The METER, which performs measurements and aggregates the results of those measurements;

2) The COLLECTOR, which is responsible for the integrity and security of METER data in short-term storage and transit;

3) The APPLICATION, which processes/formats/stores METER data. APPLICATIONS implicitly manage METERS.

The ETSI specifies interoperability mechanisms, related parameters, and quality of service in order to enable multimedia communications between circuit switched networks (such as PSTN, ISDN, and GSM) and Internet Protocol based networks [ETSI321].

The number of billing software and system solutions for ISPs is enormous [STILLER]. It is not in the scope of the AQUILA project to validate all the solutions and to look how they fit together with the AQUILA architecture or to investigate a complete new solution [FANKH]. The billing software and system offer more or less different solutions for [MINDCTI]:

- Flexible Rating
- Real-time Billing and Customer Care
- Multiple Business Models
- Prepaid and Post-paid Services
- Web-based Customer Management and Customer Self-care
- E-payments
- Customised Invoices and Reports

3.1.6.2 Flow metering of the usage data

Charging provides a feedback to the users and providers. The AQUILA architecture just offers a feedback mechanism for the user. The logged usage data is a basic for the charging and accounting of each flow. The logging data is stored in a file and the ISP can use the stored data for a post processing. For the provider is no mechanism foreseen to get accounting data directly. Each ISP can perform the pricing and billing with his own software or system solution. The AQUILA architecture only performs the groundwork for the accounting and charging of each flow.

3.1.6.2.1 Usage data for the user

The EAT is the reservation entity in the AQUILA architecture for the user. The reservation for each flow of a user is stored in the connected ACA and performed in the Edge Device. The presence of a flow allows a router to filter and count packets that belong to a certain reservation, normally a service class that is distinguished from the usual best effort packets in terms of resources required and of prices for these resources.

The ACA stores all the information of the flow and can collect the monitoring data of the router for each flow.

The retrieving method of the usage data of each flow uses the monitoring functions of the Edge Device. The router stores the monitored data in a Management Information Base (MIB). With SNMP or some commands of the CLI these can be retrieved. The router knows each flow only on the ingress side. On the egress side the flows of each traffic class are aggregated. On the ingress side the router filters and polices each flow. The usage data rest upon the monitored data of the filtering and policing.

3.1.6.2.2 Usage data *for the provider*

The cumulated statistic data of each flow will be stored in a file. The provider can us this file for a post processing with his own billing software. In the file the usage data for each user is stored. The stored parameters for a flow are:

- User ID, Session ID
- Start time, duration, end time of the reservation
- Start time of the session, login time
- Traffic Class, Traffic Spec
- Number of packets, conformed, exceeded
- Average packet rate, burst size

The format of the statistic data differs from router type to router type. The adaptation to different router types performs the EDAdapter. Therefore the format of the usage data file is independent of the router type.

3.2 Inter-domain resource allocation

Resource control between domains is generally different from resource control within a domain:

- For intra-domain resource control, there may be some entity, which has an overall picture of the domain. In the AQUILA approach, this is the RCA. However for inter-domain resource control, there will be no similar entity, which manages an overview of the whole internet.

- Intra-domain resource control has to police single flows as they enter the network in order to guarantee QoS for each customer and to block malicious flows. Inter-domain resource control however has to operate on aggregated flows in order to be scalable.

- Within a domain, one can assume a homogeneous network architecture. Intra-domain resource control however has to cope with different kinds of networks in terms of technology, routing, QoS control and scale (ISP or backbone).

These facts require a basically different architecture for inter-domain resource control than it is used within a domain. However, interoperability of resource control at both levels has to be guaranteed. So the following principles are applied, when selecting an inter-domain resource control architecture for the second phase of AQUILA:

- Inter-domain resource control is performed by an additional layer built on top of the intra-domain resource control. The interface between these two levels of resource control must be standardised and independent of the actual implementation of the intra-domain resource control, so that network operators are free to use any kind of resource control within their domain.

- Inter-domain resource control is performed by a set of independently managed, but co-operating entities, which may be called “bandwidth broker”. A protocol for communication between these entities must be specified and standardised.

The architecture specified in this document is based on these principles. Experiences from the intra-domain resource control are re-used, where applicable.

3.2.1 State of the art

Currently, mainly two approaches are available for inter-domain resource control:

- SIBBS, a protocol specified by the QBone Signalling Design Team in the context of the Internet2 project.

- BGRP, an inter-domain resource control framework developed by P. Pan et al. at the Columbia University, department of computer science.

Both approaches are shortly discussed and rated in the following.

3.2.1.1 SIBBS

The QBone bandwidth broker architecture and the SIBBS protocol are specified in [SIBBS]. This is a living document, as the QBone Signalling Design Team is still working on the architecture and the protocol.

The basic idea of SIBBS is to form a single layer of bandwidth brokers, which control the resources within each domain. Resource allocation requests from end-systems are sent to the bandwidth broker of that domain, which performs admission control for that domain and forwards the request to the next hop domain on the way to the sink domain. For communication between the bandwidth brokers, the SIBBS protocol is proposed.

The following figure is taken from the document cited above.

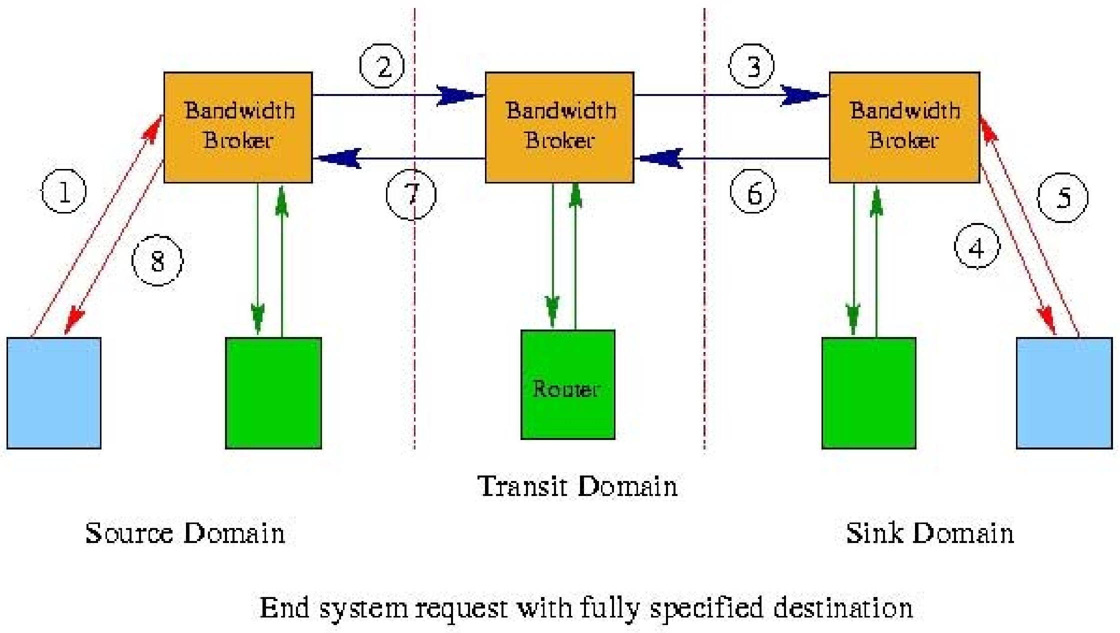

Figure 3-22: SIBBS: architecture and message flow

From the viewpoint of the requirements listed above, the SIBBS architecture has several drawbacks:

- Intra-domain and inter-domain resource control are not clearly separated. Instead, the bandwidth broker is responsible both for controlling resources within the domain and for inter-domain resource allocation.

- This makes it much more difficult to achieve a scalable solution, because the bandwidth broker has to cope with each single request.
- In fact, SIBBS basically assumes, that each end-system request is forwarded along the path to the destination domain. To reduce the number of signalling messages, core tunnels are proposed, which provide a pre-reserved pipe for further bandwidth allocations.

3.2.1.2 BGRP

BGRP is a framework for scalable resource control [BGRP]. It assumes, that BGP is used for routing between domains (autonomous systems, AS).

The basic idea of BGRP is the aggregation of reservations along the sink trees formed by the BGP routing protocol. It is a characteristic of the BGP routing protocol to forward all packets for the same destination AS to the same next hop AS. This property guarantees the formation of a sink tree for each destination AS. All traffic destined for the same AS travels along the branches of this tree towards the root.

Similar to the QBone approach, some kind of “bandwidth broker” is established in each domain. However, not just a single entity is responsible for the whole domain. Instead, an BGRP agent will be associated with each border router. Reservations for the same destination AS are aggregated at each BGRP agent. This has the following implications:

- The number of simultaneous active reservations at each domain cannot exceed the number of autonomous systems in the internet.
- The source and destination addresses cannot be carried in the reservation requests between domains, because of the aggregation mechanism.

However, the aggregation mechanism does not automatically reduce the number of signalling messages. Each request may still travel end-to-end. Additional damping is necessary, e.g. by reserving additional resources in advance or by deferred release of resources.

In summary, the BGRP framework provides a possible approach to a scalable inter-domain architecture. However, the following issues have to be solved:

- Introduction of a damping mechanism as described above. The authors of [BGRP] make some proposals here. However, also the experiences from the resource pools used for the AQUILA intra-domain resource allocation are well suited to address this topic.
- Because BGRP messages not always travel all the way to the destination domain, the problem of QoS signalling within the last domain towards the destination host has to be solved.
- BGRP is still a framework only. The detailed information exchange between BGRP bandwidth brokers as well as the interaction with the intra-domain resource control have to be specified.

3.2.2 Requirements

An architecture for the AQUILA inter-domain resource control has to fulfil the following requirements:

- Scalability

  When high quality services will be established in the internet world-wide, the number of individual resource reservations will grow rapidly. The architecture must be able to cope with that.

- Works with multiple intra-domain resource control mechanisms

  Operators should be free to use any resource control mechanism within their domain. The AQUILA intra-domain approach is just one possible example. An interface must be defined and standardised, through which the inter-domain resource control interacts with the domain specific QoS mechanisms.

- Edge-to-edge QoS guarantee

  The architecture must be able to support a certain level of QoS guarantee from the ingress edge of the source domain to the egress edge of the destination domain.

- Stepwise deployment

  It must be possible to deploy the architecture in the internet step by step. An architecture, where any modification or enhancement has to be installed in each AS, is not acceptable.

3.2.3 Architecture

In order to fulfil the requirements listed above, an architecture according to the BGRP framework will be chosen. However, a number of extensions and enhancements have to be added to make a running implementation out of the framework.

Also, ideas and mechanisms developed for the intra-domain resource control will also influence the AQUILA inter-domain architecture.

This chapter specifies the general architecture for the AQUILA inter-domain resource control, where the next chapter addresses detailed aspects.

The following picture gives a rough overview of the architecture and depicts the basic interactions between the intra- and inter-domain resource control layer in the source, intermediate and destination domain.

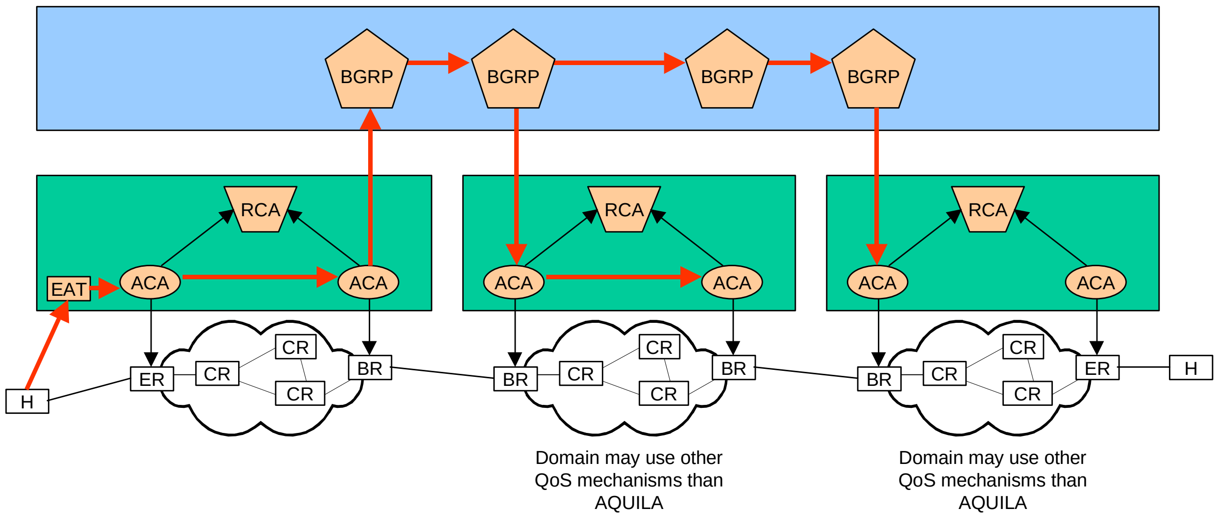
BGRP
BGRP
BGRP
BGRP
RCA
RCA
RCA
EAT
ACA
ACA
ACA
ACA
ACA
ACA
H
ER
CR
CR
CR
BR
BR
CR
CR
CR
BR
BR
CR
CR
CR
ER
H
Domain may use other QoS mechanisms than AQUILA
Domain may use other QoS mechanisms than AQUILA

Figure 3-23: General inter-domain architecture and message flow

A so-called BGRP agent is associated with each border router. These agents interact with the AQUILA intra-domain resource control layer in the following way:

- Inter-domain resource requests are initiated by the ACA associated with the egress border router of the initiating domain and sent to the corresponding BGRP agent.
- BGRP agents associated with ingress border routers use the ingress ACA to establish intra-domain resource reservations.

3.2.4 Detailed aspects

3.2.4.1 Offered services

When network services spanning multiple domains administered by different operators are offered, there must be some common understanding of these services between the domains. The offered services also have to be known to the customer. To accomplish this, two opposed approaches are possible:

- Globally well known services (GWKS): There is a set of standardised services, which are implemented by all domains. The customer also knows this set of services he/she can select from.
- Dynamic detection of services: The network may have the ability to detect the possible services from a given source to a given destination which arise from chaining of different services in the different domains on the path from the source to the destination.

The latter induces some serious problems, which largely reduces its usability: When each domain may define its own network services, some kind of mapping has to be performed at each domain boundary. When multiple mappings are chained, it is very likely that the resulting network service is very unspecific and not usable for the end-user's application.

Another problem arises, when one of the domains defines fewer network services (e.g. just one premium service) than the previous one and subsequent domains. In this case, several network services have to be mapped to a single service, when that domain is entered. Later on, these packets can no longer be separated in different services in subsequent domains, which possibly again implement a larger number of network services.

So it seems to be necessary, that a fixed number of globally well known set of network services is defined, so that the mapping at the domain boundaries can be eliminated. For each GWKS some parameters of the traffic description are bound to a fixed range, e.g. a maximum packet size.

Network operators may however choose to implement these GWKS to different extends, which may be chained. E.g. if there is one GWKS, which is tailored for low delay, then each network operator may choose to guarantee different values for the mean and the maximum delay. The delay values may be added along the path from the source to the destination to give the overall delay. In a similar way, overall path packet loss ratios or other QoS parameters may be computed from the values advertised by each domain along the path and presented to the end-user.

So the following approach for the definition of network services is proposed for AQUILA:

- Define a set of globally well known services. However, instead of fixing all QoS parameters for these services, leave them open and specify an optimisation target, e.g. no loss, very short delay, etc.

- When an end-user requests a network service, he also specifies a set of QoS parameters, e.g. a maximum delay he is willing to accept. These QoS parameters are sent as part of the request message along the path to the destination domain. Each hop updates this field to take into account the delay, jitter and packet loss introduced by the path segment from the previous hop to the current location. A request is rejected, if the path cannot satisfy the end-users requirements.

Specification of the path segment QoS values might be part of a service level agreement between neighboured domains. Note, that it is important, that each network operator gives a honest value. One should neither promise more, than what can be kept, nor specify too unfavourable values, just to be “on the safe side”.

Note also, that inter-domain services need to specify both an ingress and an egress point (source and destination address). Point-to-any services as the PMC service specified in [D1301] cannot be used in an inter-domain scenario, because it is unknown, which domains the traffic will cross and therefore one is unable to give guarantees for that traffic.

3.2.4.2 Sink-tree-based aggregation

In the internet, BGP is used as the inter-domain routing protocol. The border routers are the hops, which are considered by BGP. It is a property of the BGP routing protocol to create paths in such a way, that a BGP router forwards all packets to the same destination via a single next-hop router. This guarantees the creation of so-called sink trees. All traffic to a specific destination travels along the branches of such a sink tree towards the root of the tree, which represents the destination.

The AQUILA inter-domain reservation architecture uses this property. It forwards path discovery messages (PROBE) along the BGP routing paths. Reservations are set up in the reverse direction (GRAFT). At each BGP hop (border router), reservations for the same destination (i.e. belonging to the same sink tree) are aggregated.

Note, that this primarily limits the number of simultaneous reservations to be kept at each border router. However, additional mechanisms have to be used to reduce the number of signalling messages travelling end-to-end (see 3.2.4.6, Damping mechanism, page 60).

3.2.4.3 Co-operation with BGP

To determine the path towards the destination domain, the inter-domain resource control layer has to interact with the BGP routing, to get the next hop BGRP agent. The IP address of the next hop border router is determined from the NEXT_HOP attribute of the BGP route.

The only interface between BGRP and BGP will be a possibility for BGRP to query

- a list of neighbours to establish the communication channels between the BGRP agents;
- the BGP NEXT_HOP for a given destination address;

3.2.4.4 BGRP agent communication

BGRP agents are set up for each border router in an BGRP enabled domain. Communication is always between “adjacent” BGRP agents. Adjacent BGRP agents can be in the same domain or in different (neighboured) domains.

Between each pair of adjacent BGRP agents, a reliable and secure communication channel is established.

Reservations are always initiated at the BGRP agent associated with the egress border router of the source domain. This agent represents a leaf in the sink tree towards the destination domain and is therefore called the leaf BGRP agent of the reservation.

The end-point of a BGRP reservation is the BGRP agent associated with the ingress border router of the destination domain. This agent represents the root of the sink and is therefore called the root BGRP agent.

In transit domains, the BGRP agents associated with both the ingress and the egress border router are involved in processing a request. These agents are called ingress and egress BGRP agents.

3.2.4.4.1 Reservation set-up

To set-up a reservation, basically two messages are involved: PROBE and GRAFT.

The leaf agent initiates a reservation by sending a PROBE message to the next hop towards the destination. The PROBE message contains the reservation request, a set of QoS parameters, destination network information and a route record. PROBE messages travel downstream from the leaf agent to the root. When a PROBE message arrives at an ingress BGRP agent, that agent consults its resource database to verify, whether the new reservation fits into bilateral agreement with the previous domain. It also checks, whether the requested QoS parameters can be satisfied up to that location. If the new reservation can be accepted, the BGRP agent inserts its own IP address into the route record field and forwards the PROBE message to the next hop. Note, that no reservation is carried out and no state is kept in any of the involved BGRP agents during the PROBE phase.

PROBE messages are terminated either due to an error condition at an intermediate BGRP agent or when the message reaches a BGRP agent, which can doubtless identify the tree identification for the destination network (see 3.2.4.7, Quiet grafting, page 61). At the latest, the root agent can terminate the PROBE message.

When the PROBE message successfully arrived at the terminating BGRP agent, a GRAFT message is sent back along the path recorded in the route record field of the PROBE message. The GRAFT message contains a tree ID, which uniquely identifies the sink tree.

At each hop, the reservation is merged with other reservations for the same sink tree. At each ingress hop (except the one in the destination domain), local reservations are carried out to request resources within domains.

GRAFT messages also contain a reference to the intra-domain resource control of the last domain along with the IP address of the ingress border router, which together can be used by the initiating domain to request resources in the last domain along the path from the ingress border router to the end-user host (see 3.2.4.8, Signalling in the last domain, page 62).

3.2.4.4.2 Refresh

Reservations are periodically refreshed between peers. For this purpose, a BGRP agent periodically sends an UPDATE message to all known next-hop and all known previous-hop agents. The UPDATE message contains a list of all active reservations for that hop.

The UPDATE for a large number of reservations may be sent in several messages, provided, that the timeout is not exceeded for each reservation .

If an agent does not receive an UPDATE message for a reservation for a specific amount of time, it assumes, that the affected reservations are no longer active and removes them from its internal tables. If an UPDATE message from a previous-hop agent specifies a lower amount of resources than stored in the agent's internal tables, it assumes, that the reservation was reduced. If an UPDATE message from a next-hop agent specifies a higher amount of resources than stored in the agent's internal tables, it assumes, that this higher amount is currently reserved by the next hop. The agent may send a TEAR message to affected next-hop agents to release resources no longer used. However, this is not mandatory. The agent may also wait for the UPDATE mechanism to propagate the released reservations further through the network.

If an agent receives an UPDATE message with more reservations or reservations with a higher amount of resources than stored in its internal tables from a previous-hop agent, this is considered to be an error. It is also an error, if an agent receives an UPDATE message with less reservations or with a lower amount of resources than stored in its internal tables from a next-hop agent. To recover from this error, the agent will try to request the additional resources from the next-hop agent by immediately sending an UPDATE message with the required amount of resources for this reservation to the next-hop agent. This UPDATE message will contain a "please confirm" option, which forces the receiver to answer the request either with an UPDATE or an ERROR message within a certain amount of time. If an ERROR message is received or a timeout occurs, the agent will send an ERROR message upstream to all affected previous-hop agents.

3.2.4.4.3 Reservation tear-down

To release a reservation or to reduce the amount of resources allocated to a reservation, the refresh mechanism described before may be used. However to speed up the operation, a TEAR message may be sent. A TEAR message may be seen as a partial UPDATE message. Each agent is free to release the resources as a whole or partially from the next hop agent with a TEAR message, to wait for the UPDATE mechanism, or to keep the resources for later use.

A TEAR message with a zero amount of resources will remove all information for that reservation from the agent's internal tables. When no more previous-hop entries exist for a given reservation, an agent will also send a TEAR message to the next-hop agent.

3.2.4.4.4 Errors during reservation

During the PROBE phase an agent may detect, that the QoS parameters requested for that reservation cannot be handled by the network. An agent associated with an ingress border router may also detect, that the additional request cannot be covered by the bilateral agreement with the previous-hop network operator. In these cases, an ERROR message may be sent back already during PROBE processing.

During the GRAFT phase, an agent may detect, that resources cannot be reserved for that part of the path it is responsible for. In this case, the agent will send an ERROR message upstream to inform the requestor and may additionally send a TEAR message downstream in order to release the already reserved resources. Sending of the TEAR message is optional. The agent may also wait for the UPDATE mechanism to propagate the change or keep the resources for possible future reservations.

3.2.4.5 Interface to intra-domain resource control

An inter-domain reservation is always initiated by some intra-domain resource control entity in the source domain. Inter-domain resource control uses an intra-domain resource control entity to request resources within a domain and on links between domains. Figure 3-22 on page 50 illustrates the communication between these entities.

The interface has to fulfil the following requirements:

- At the initiating domain: Request of resources to a destination domain. The request should contain the network service, traffic descriptor, required QoS parameters, destination. The response should contain a success indicator and the actual QoS parameters.
- At a transit domain: Request of transit resources through a domain. The request should contain the network service, minimum traffic descriptor, preferred traffic descriptor, ingress and egress. The response should contain a success indicator and the actual QoS parameters.
- At the destination domain: Request of destination resources. The request should contain the network service, minimum traffic descriptor, preferred traffic descriptor and the ingress. The response should contain a success indicator and a reference ID, which can be used by the intra-domain resource control of the initiating domain to request resources within the last domain from the ingress to the end-user host.

3.2.4.5.1 AQUILA intra-domain roles

In [D1201], AQUILA defines a three-role-model for intra-domain reservations. The three roles are:

- a requestor, which initiates the request;
- a sender, which injects the traffic into the network,
- a receiver, which receives the QoS enabled traffic.

The inter-domain approach specified in the current document however, follows a sender-initiated model for reservations. This is necessary, because the GRAFT messages for path discovery always have to follow the BGP routing path from the sender to the receiver. This path cannot be discovered in the reverse direction from the receiver to the sender, nor can it be determined by a third party.

To match both approaches, the following mechanism for three-role-reservations is proposed:

1. If the requester is not in the same domain as the sender, the requester will initiate a reservation to the sender with a bandwidth request of zero. The answer to this request will include the reference to the intra-domain resource control of the sender domain.

2. The requester will then request the required resources from the intra-domain resource control of the sender domain. This request will include information, which enables the sender domain to charge the requester for the reservation.

3. Intra-domain resource control of the sender domain will then carry out a "normal" sender-initiated resource request, including the initiation of the inter-domain request to the receiver domain, if necessary.

This approach to the three-role-model adds the following requirement to the interface to intra-domain resource control:

- At the source domain: Request of inter-domain resources. The request should contain the network service, requested QoS parameters, traffic descriptor, source, destination and charging authorisation information. The response should contain a success indicator and the actual QoS parameters.

3.2.4.5.2 Role of ACA for border routers

In the intra-domain architecture of AQUILA, admission control agents are responsible for traffic carried over edge devices into and out of the network. In a multi-domain scenario, border routers play a similar role for traffic exchanged between neighbouring domains. There is, however, one mayor difference:

While edge devices are able to police and mark traffic on a per flow base, this is not achievable at border routers. So border routers have to police traffic on a coarser level. There are the following two possibilities:

- Border routers may police and mark traffic "per reservation". In the BGRP scenario this means per sink tree and per traffic class.

- Border routers may police and mark traffic "per traffic class", aggregating all reservations for the same traffic class.

Note, that policing at domain boundaries has a very different basis than policing at the ingress edge device:

- Policing at the edge device controls, that the customer does not send more traffic than admitted. Excessive traffic may be degraded or dropped. The network operator does not give any guarantee for that.

- Policing at the border router however controls, that the neighbour domain does not send more traffic than admitted. However, if you choose to drop or degrade the excessive traffic, the end-customer will be affected, not the neighbour domain. So dropping is not the right "punishment" for excessive traffic between domains.

Instead, it might be useful to negotiate service level agreements between domains in such a way, that excessive traffic is charged with a high price. In this case, the neighbour domain will be interested in controlling the QoS traffic it generates or receives and forwards. However, the definition of such models is behind the scope of the project.

In any case the border router has to police incoming traffic. This might be done on different levels of detail:

- Per reservation handling:
  - Border routers must be able to set up a multi-field classifier based on the DSCP field (to identify the traffic class) and a list of IP prefixes (to identify the sink tree). Currently, the AQUILA intra-domain architecture can handle only a single IP prefix.
  - When the neighbour domain sends more traffic than admitted, the policer can identify the violating reservation.
- Per traffic class handling:
  - Border routers must be able to set up a classifier based on the DSCP field.
  - The policer cannot detect excessive traffic in a single sink tree, as long as the sum of the traffic in all trees of the same traffic class does not exceed the overall limit.

For the AQUILA project we propose to use per traffic class handling. The policer should be set up in such a way, that excessive traffic is not immediately dropped or degraded, but reported for further administrative actions. Dropping or degrading should occur, when the traffic reaches a level, which would affect the network's ability to carry other (non-offending) QoS traffic.

3.2.4.6 Damping mechanism

Cited from [BGRP]:

> *The basic BGRP protocol aggregates reservations into trees, thereby reducing the number of reservations. We will quantify this in Sections V-A and V-B. Reducing the number of reservations obviously shrinks the memory needed to store the control state information. It also reduces the overhead associated with REFRESH messages for all these pieces of control state; refresh costs include CPU processing and link bandwidth. These savings take us much of the way toward our goal. However, BGRP's other control messages, PROBE and GRAFT, also consume processing and bandwidth. We would like to control the volume of these control messages as well and thereby add another dimension of scalability to BGRP. This can be done by making the following enhancements to the protocol.*

The enhancements proposed by the cited paper include the following:

- Over-reservation, quantisation and hysteresis
- CIDR labelling and quiet grafting
- Reservation damping

The first proposal tries to reduce the number of reservation messages by using mechanisms, which request more resources than actually needed or delay the release of resources, in order to use them for subsequent reservations. Note however, that this does not reduce the number of PROBE messages. This proposal is discussed in more detail later on in this chapter.

The second proposal also addresses PROBE messages. It tries to respond to PROBE messages already at an earlier point in the sink tree, by enabling intermediate BGRP agents to identify the sink tree, which will be used for that reservation. This topic is discussed in more detail in chapter 3.2.4.7, Quiet grafting, on page 61.

The third proposal addresses the problem of reservation changes due to BGP route changes. As many of the daily BGP route changes are pathologically and do not reflect real network topological changes, mechanisms could be used to reduce the number of unnecessary reservation moves. In spite of the fact, that this is a undeniable problem in real networks, the project will not further investigate it.

3.2.4.7 Quiet grafting

So far, the source domain cannot uniquely identify the destination domain, and thus the sink tree for a new reservation, without sending the PROBE message all the way to the destination domain. BGP does not provide this information in its AS_PATH attribute, because paths leading to a set of autonomous systems may be aggregated and can no longer be distinguished from the source domain's point of view.

Quiet grafting tries to enable other nodes before the final one to determine the sink tree, to shorten the average distance the PROBE and GRAFT messages have to travel. This will also reduce the signalling load.

To enable quiet grafting, the GRAFT messages will additionally carry a (possibly incomplete) list of announced IP address prefixes of the destination domain. This list contains at least the IP address prefix covering the destination address sent in the PROBE message, but may also include other address prefixes announced by the destination domain. However, this list may not contain any IP address prefix space, which is possibly covered by another more specific route. Each BGRP agent stores this information along with the reservation. The restriction formulated in the previous sentence guarantees, that there are no overlapping IP prefixes stored in this table. If – due to any failure or disregard of that restriction – overlapping IP prefixes exist in that table, the most specific prefix should be used for any lookup in this table.

When a PROBE message arrives at a BGRP agent, the current list of known IP prefixes is consulted to possibly find an already existing reservation towards that destination domain. BGP is consulted to verify that the reservation found matches the IP prefix for that destination. If an existing matching reservation is found, pre-allocated resources may be used and the PROBE message may already be answered with a GRAFT. In addition, the offered QoS parameters from the actual location up to the destination may be compared to the requested parameters and the reservation may be rejected, if the requested QoS parameters cannot be met.

3.2.4.8 Signalling in the last domain

The damping mechanisms and quiet grafting described above inhibit the forwarding of signalling messages to the last domain. So that domain may be unaware of a new reservation. Specifically, no resources are reserved on the path from the ingress border router to the destination host.

The AQUILA inter-domain architecture addresses this problem by returning a reference to the intra-domain resource control and the address of the ingress border router within the GRAFT message to the initiating domain. The initiating domain then explicitly requests the resources in the destination domain, if necessary.

In order to authorise this request, it has to contain a reference to the bilateral agreement of the last domain’s network operator with the last but one, so that the request could be interpreted as follows: “You (the last domain) have already promised to the last but one domain to receive my traffic. With reference to this promise I request to carry my traffic to the final destination within your domain.”

3.2.5 Scalability

Scalability in resource reservation has several – partly correlated – dimensions:

- **Number of reservations:** How many reservations are simultaneously active at each network element? A high number of reservations needs a high amount of memory to store the reservation states. Additionally, this will also mean a high number of signalling messages, because the states most probably need to be refreshed regularly.

- **Number of hops a reservation request has to cross:** Can a reservation request be handled “near” to the requester or does it have to travel all the way to the destination? A large number of hops means a high number of signalling messages in the network and thus a high amount of CPU processing power, especially for backbone domains.

The proposed architecture inherently solves the first issue. The number of simultaneous reservations cannot exceed the number of autonomous systems in the internet multiplied by the number of network services offered. So the architecture exposes a linear effort growth regarding the number of autonomous systems in the Internet.

However, without the quiet grafting and damping mechanisms, each signalling message still has to travel all the way from the source to the destination domain. As shown in [BGRP], this may lead to scalability problems, especially for short-lived small bandwidth flows, such as VoIP. Transit domains are mainly affected by this behaviour.

3.2.5.1 A first approach for quiet grafting

The use of BGRP as the inter-domain protocol in the AQUILA architecture provides an end-to-end resource allocation solution. However, the current proposal of the BGRP does not contain a detailed analysis of the protocol and in addition the protocol is not yet implemented in a real network environment. That implies that the deployment of the BGRP protocol to the AQUILA architecture should be analysed detailed described and finally implemented. So, many open issues should be encountered and a first solution should be given.

This contribution handles with the scalability problems arising from the deployment of the BGRP, and especially with issues such as quiet grafting and damping mechanism. A first approach is given as far concern the quiet grafting mechanism, which provides a solution to scalability problems.

3.2.5.2 PROBE & GRAFT messages

The main scalability problem arises from the fact that a reservation request has to cross a significantly large number of hops in order to reach the destination host. The latter implies that a large number of PROBE and GRAFT messages should be exchanged and processed by the intermediate Border Routers. That has as result a lot of bandwidth and processing power to be consumed. Therefore, a major task is the control of number of PROBE and GRAFT messages, limiting their number and adding in this way scalability to the BGRP protocol.

Quiet Grafting is a basic solution for limiting the signalling messages. In [BGRP], it is not described the mechanism, but only a first approach is given.

3.2.5.3 Classless Inter-Domain Routing (CIDR) Labelling

Classless Inter Domain Routing (CIDR) is a new addressing scheme for the Internet, which allows for more efficient allocation of IP addresses than the old Class A, B, and C address scheme. CIDR is a replacement for the old process of assigning Class A, B and C addresses with a generalised network "prefix". Instead of being limited to network identifiers (or "prefixes") of 8, 16 or 24 bits, CIDR currently uses prefixes anywhere from 13 to 27 bits. Thus, blocks of addresses can be assigned to networks as small as 32 hosts or to those with over 500,000 hosts.

A CIDR address includes the standard 32-bit IP address and also information on how many bits are used for the network prefix. Therefore, routing destinations are represented by network and mask pairs and moreover routing is done on a longest-match basis (i.e. for a given destination, which matches multiple network-mask pairs, the match with the longest mask, is used). A typical CIDR address for a network could be of the form 147.102.32.0/19 where the IP address is 147.102.32.0 and the mask is 255.255.224.0 (it corresponds to 19 bits). In this way, routers do not just keep the IP address of the networks for making routing decisions but also maintain the information that is introduced by the mask. It is obvious that an IP address of the old addressing scheme, for example 147.102.240.0 is different from the same IP address of the new scheme where a mask is additionally defined, i.e. 147.102.240.0/20. As a result, CIDR brings flexibility and adaptability to assigning address spaces that fit an organisation's specific needs.

It is often the case that the border routers are not aware of the exact set of CIDR addresses that are reachable to a specific border router. This is due to the fact that the CIDR addressing scheme enables "route aggregation" in which a single high-level route entry can represent many lower-level routes in the global routing tables. Therefore, a border router may keep the list of its reachable CIDR address destinations but will not advertise the same list to the adjacent border routers. It will perform an aggregation of its routes if possible (it will not aggregate those routes that cannot be treated as part of a single unit due to multi-homing, policy or other constraints) [RFC1519] and will advertise the aggregation of routes that will always be a superset of the explicit routes.

It is envisaged that CIDR will be used in BGRP for the labelling of the sink tree. Therefore, an analysis is needed for the identification of the possible problems arisen and the enhancements introduced to the scalability of BGRP.

Since CIDR labelling will be used for the identification of a reservation tree, it is important that a CIDR address suffices to guarantee the uniqueness of a sink tree passing from a border router. In other sense, it is required that a router can tell, based only on the CIDR label, that a new reservation belongs to a sink tree and there is no way that there is another sink tree with the same CIDR label passing from the same router. For example, it is possible that two border routers of an AS reach the same sub-domain and therefore advertise it as one of their destinations. Both routers can form sink trees for this destination domain (they are roots of the sink trees) making in this way the CIDR labelling not sufficient for identifying a sink tree. It is obvious that the IP address of the router is also needed for that purpose.

Nevertheless, taking into consideration that the BGP route selection is based on the shortest path, it is certain that there is only one route passing from a border router to a specific destination. Therefore, the above assumption is not valid and the uniqueness of a CIDR labelled sink tree is guaranteed within the list of sink trees that a border router belongs to. The following figure (Figure 3-24) depicts more clearly the above concept.

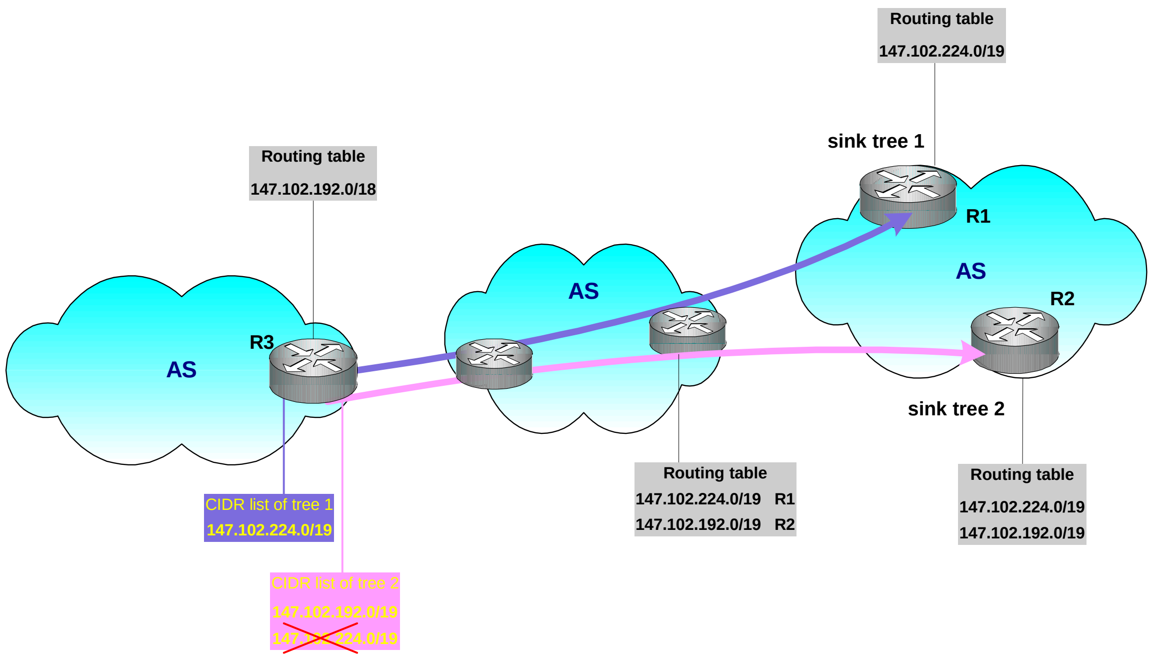

Figure 3-24-Uniqueness of a CIDR labelled sink tree

Packets passing from R3 to destination 147.102.224.0/19 will always be forwarded to router R1 based on the shortest path algorithm that is used by BGP (even if this destination is also reachable by R2). Therefore, they will belong to tree 1 and as a result the CIDR label of each tree will be unique. In case that R2 also advertises the same destination to the R3 BGRP agent, the agent has to understand that this label is not valid since the packets to that destination will not follow the tree 2 route based on the BGP routing decisions.

As mentioned before, CIDR addressing scheme performs route aggregation. Therefore, a border router may not be aware of the explicit routes of an aggregation but will most often know the summarisation of these routes. Supposing, that two border routers (R1 and R2) of an AS have access to the CIDR addresses 147.102.224.0/19 and 147.102.192.0/19 respectively. A distant router may not be aware of that explicit information but might instead be aware of a route to 147.102.192.0/18 CIDR address (aggregation of the two addresses). Supposing that one of these routers (R1) is root to a sink tree and that the distant router (R3) belongs to that tree. It is therefore essential that the latter has a more specific view of the domains that can be reached by the sink tree it belongs to enabling in this way the quiet grafting of a future reservation request to the tree. Therefore, it is required that the root of a sink tree provides the other border routers with the list of its reachable sub-domains (within the GRAFT message). This list should at least contain the destination CIDR address of the existing resource reservations, which implies that it will be continuously updated while reservations for new destinations are incorporated into the sink tree. Apart from the CIDR destination addresses of the existing reservations, the root can also include in the GRAFT message some more reachable CIDR addresses in order to enable quiet grafting for future reservations to new destinations. It is FFS whether or not these destinations are included at the first GRAFT message that forms the sink tree or they are gradually distributed to the routers of the tree.

Moreover, a certain problem arises from the advertisement of the whole CIDR list from the root of a sink tree. Supposing that the border router R2 is also root to a sink tree to which R3 belongs as well. One more assumption is that R2 can also reach the sub-domain 147.102.224.0/19 and that it has sent this information to R3 within the GRAFT message. Therefore, router R3 maintains two lists of CIDR addresses for the two different sink trees. The list that has been created from the sink tree of R2 contains two CIDR addresses, 147.102.224.0/19 and 147.102.192.0/19 whereas the other list contains the CIDR address 147.102.224.0/19. At this point, when a new reservation request is made for the domain 147.102.224.0/19, the BGRP agent that resides at the router R3 will not know the sink tree to which it should graft the current request. One possible solution is that it keeps some further information in order to distinguish between the CIDR addresses of a list (whether there exist reservations for destination CIDR addresses). In this way the BGRP agent will know that the sink tree of the router R1 provides the only route to that destination (if a reservation has already been made to that tree for that destination). The above problem is illustrated in the following figure (Figure 3-25).

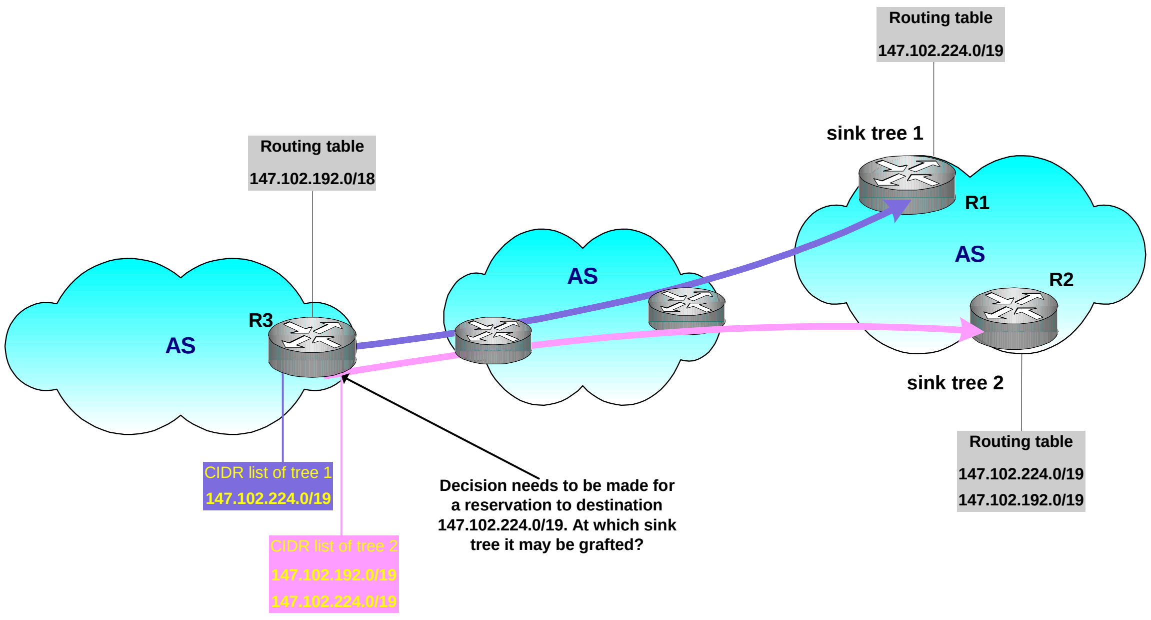

Figure 3-25-Distribution of destination CIDR addresses

It is important to be noted that CIDR is supported by BGPv4 and implemented in most routers.

To conclude, the CIDR labelling guarantees the identification of the sink trees that a border router participates in and therefore justifies its use for the tree label. Furthermore, the way that a root of a sink tree distributes its destinations as well as which destinations are finally distributed has to be decided.

3.2.5.4 Mechanism for Pre-Allocating Bandwidth

As aforementioned, a certain amount of bandwidth should be pre-allocated in order to serve future requests, and in this way limiting the number of PROBE and GRAFT messages.

Resource Pools algorithm for dynamic bandwidth distribution might be a first solution, but the BGRP is a particular case with different requirements and characteristics. The main factors that influence the algorithm are:

- The arrival rate of reservation requests
- The duration of each reservation
- The average bandwidth of reservations requests
- The sink trees created
- The popularity of a certain sink tree (destination host)
- The target network resources utilisation
- The target limitation of messages overhead

All those factors should be considered when determining the algorithm for pre-reservations.

The BGRP agent will not be regarded as a part of the RPool hierarchy, but it has partially a common functionality with the RPools.

3.2.5.4.1 Mechanism for Releasing BW: A crucial Point

The concept of quiet grafting is based on a resource pre-allocation mechanism in order to serve future requests. By following that concept the risk of reserving resources that aren't actually needed is emerging.

Thus, an additional mechanism for the release of any "unwanted" pre-allocated resources should be defined. The term "unwanted" is used for resources that have been pre-allocated but based on some criteria are not needed any more. These criteria might be defined following many different approaches (i.e. time-based, event-based or measurement-based).

In a time-based approach the pre-allocated resources will be released if they haven't been used after a certain period of time. That approach requires the use of timers in order to keep track of the time period, which generally might add complexity in the system. However, this is not our case because timers will be included for the support of the UPDATE messages. The event-based approach is actually the one that is currently used in AQUILA. Each time a reservation is released, the RPL is checking whether unused resources should be given back or not. Each time an event occurs (e.g. release of a reservation) an algorithm, which can be similar to those that are already discussed in AQUILA, will determine if there are any "unused" resources and how many of them should be released. Finally, the measurement-based approach could use measurements concerning the utilisation in order to take the decision for the release of resources.

Nevertheless, the approach that will be chosen should be consistent with the logic of the pre-allocation mechanism. In other words, the criteria used for the pre-allocation of resources should be conformant to those for the release otherwise the mechanism will become inefficient. For instance, if the criterion for pre-allocating resources is the rate of the reservation requests (which might reflect the popularity of the given sink tree) then more resources will be pre-allocated to that node. If, on the other hand, the release algorithm is invoked every time a reservation request is received, that node will very frequently check for “unwanted” resources while probably all the assigned resources are “wanted”.

The actual specification of the algorithms used in AQUILA is contained in deliverable D1302.

3.2.6 Analysis class diagram

Taking into account the requirements and issues discussed in the previous sub-chapters, the following architectural class diagram describes the BGRP based inter-domain resource control. It does however not cover any interfaces or interaction with intra-domain resource control.

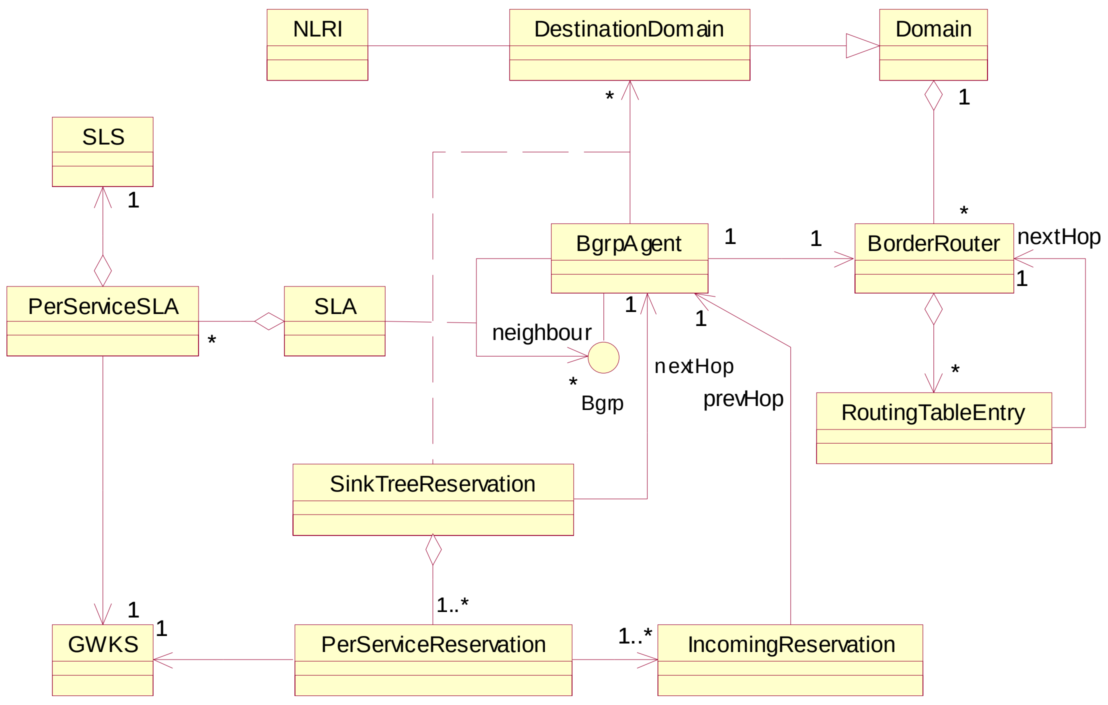

Figure 3-26: Architectural class diagram of BGRP resource control

We now describe the objects and relations starting from the **Domain** in the upper right part of the diagram. Terms referring to one of the classes shown in the diagram are printed in bold: Each **Domain** consists of (amongst other objects not shown here) a set of **Border Routers** running BGP. The main task of these routers is to use their **Routing Table Entries** to determine the next hop, which is (in the BGP world) again a **Border Router**.

Now a **BGRP Agent** is instantiated for each **Border Router**. A **BGRP Agent** implements a **BGRP** interface, which is used by all neighboured **BGRP Agents** to communicate the BGRP protocol messages. An **SLA** is attached to each neighbour association. An **SLA** contains (amongst other objects not shown here) a set of **Per Service SLAs**, each associated with a specific **GWKS**. **Per Service SLAs** contain (amongst other objects not shown here) e.g. an **SLS** containing technical parameters.

BGRP Agents manage **Sink Tree Reservations** for each **Destination Domain**. A **Sink Tree Reservation** consists of one or more **Per Service Reservations**, which are associated with a specific **GWKS**. A **Sink Tree Reservation** knows a single next hop, to which all packets travelling over this **Sink Tree** are forwarded.

A **Per Service Reservation** is the result of at least one **Incoming Reservation**, which was received from a previous hop **BGRP Agent**.

The inter-working with intra-domain resource control is depicted in the following diagram:

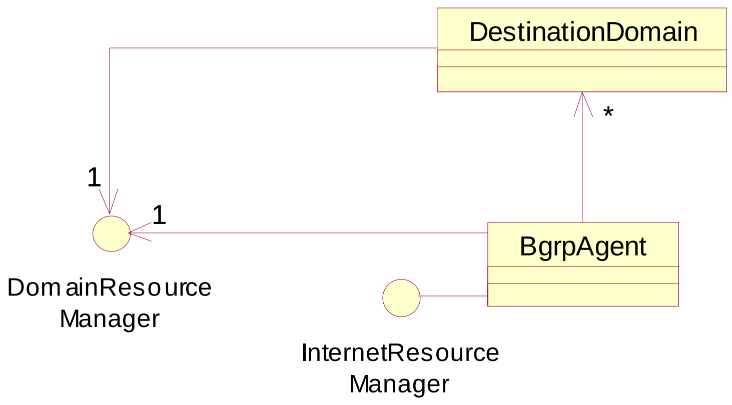

Figure 3-27: Inter-working with intra-domain resource control

The **BGRP Agent** implements an **Internet Resource Manager** interface, which is used by the intra-domain resource control or some other entity to request reservations. The **BGRP Agent** uses an interface called **Domain Resource Manager**, which provides the necessary intra-domain resource control. The agent also stores a reference to a **Domain Resource Manager** interface per **Destination Domain** in order to provide means for signalling in the last domain (see chapter 3.2.4.8). The definition of this interface should not be AQUILA specific, but also support other types of intra-domain resource control. In the AQUILA project, however, the ACA will implement this interface.

3.3 Application interfaces

3.3.1 Non QoS-aware application support

One of the aims of the AQUILA project is to support legacy applications without changing them. The AQUILA project proposes an approach based on application profiles and proxies. The application profile approach can be used if precise information about an application is known, the proxies are in a first time used when an application uses for its data flow dynamically negotiated ports. The application profile is used by a converter that translates end-user decisions at application level into reservation requests.

One aim of the second trial is to develop a complex Internet service that will use the capabilities of the AQUILA solution.

3.3.1.1 State of the art, references, ideas

There are few QoS projects taking (legacy) applications and end-users into account. The following chapter gives an introduction to the different approaches relevant for the AQUILA approach.

- The project SABA2 [SABA2] supports multimedia non QoS-aware applications with static (VIC and VAT) and variable (ISABEL [ISABEL]) QoS requirements. It proposes a RSVP [COMA] agent based on a generic QoS API [QOSWG] that monitors the traffic generated by the application. The RSVP agent generates RESV messages following the transmission rate of the application.

- In the works [BISSEL], [BOGEN] interactive real-time applications are analysed in a comparable way as for the AQUILA application profiles. This work could be a further information source.

- The INDEX project [ALTM] is indirectly for relevance for the AQUILA application support approach. This project gives arguments for an user friendly approach for providing QoS.

  "The INDEX project, which stands for INternet Demand EXperiment project, is a field experiment for investigating people's willingness to pay for a certain service quality. The subjects of the INDEX project, who got a installation of a dedicated ISDN line between their homes and the Internet for free, have the possibility to choose between different options (i.e. as connection speeds) at different prices depending on the current experiment running."

  „Since the user is concerned about how much money he is going to spend on a service, this item has to be taken into account when designing a QoS architecture." [BRUS]

- Here a short description of the Eclipse [BLAN] operating system. This is for information only. The methodology used is in our opinion not relevant for the AQUILA approach.

Nevertheless a reviewer of a submitted paper to the IWQoS2001 mentioned it as missing reference.

The Eclipse operating system is a testbed for Quality of Service (QoS) based on FreeBSD version 3.4. Eclipse provides flexible and fine-grained Quality of Service (QoS) support for applications. Eclipse is being used to guarantee QoS to server applications, in particular, to differentiate the performance of different web sites hosted on the same platform. Have been implemented:
i) hierarchical proportional-share CPU, disk and link schedulers,
ii) the /reserv file system providing an API to manipulate "reservations" and
iii) a tagging mechanism for the association of reservations with schedulable operations.

"*The use of default lists and garbage collection makes it possible to provision resources for legacy applications. Reservations can be assigned to the default list of an application without its knowledge and it will transparently obtain the QoS support provided by these reservations. When the application finishes, these reservations can be transparently garbage collected.*"

3.3.1.2 Application profiles and basic applications

The aim of the profile is to enable and keep a mapping activity between the end-user, the application and the network transparent. For the moment the profile support the QoS offer towards end-user for legacy (non QoS-aware) basic applications. A basic application is a usual Internet component like: voice over IP application, video conferencing application, TV on demand, ftp, streaming application, audio applications...

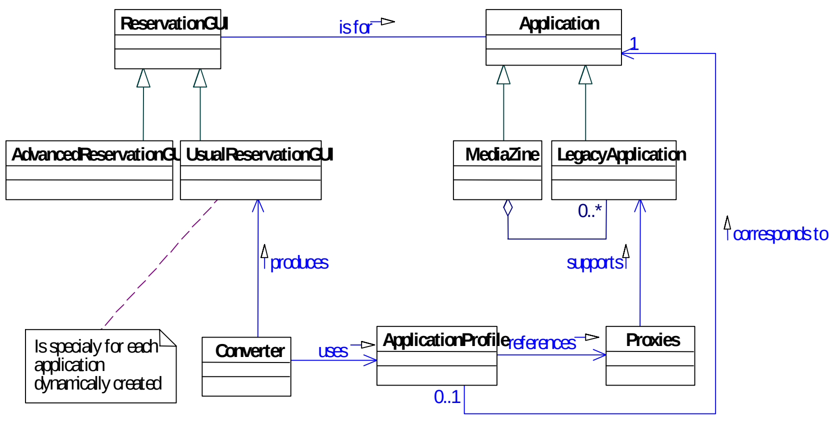

Figure 3-28: Relation between the reservation GUI and the application profile

The current profile describes the information needed for the mapping between application level and the network level and is very close to the AQUILA architecture and approach and not very generic. It helps to transform the end-user request in an AQUILA request. A converter is responsible for this mapping.

An application profile can be only used if an analysis of the targeted application has been done.

The information structure is ruled by a DTD and saved in a corresponding XML file.

3.3.1.3 Application profiles refinement

The current version of the profile structure has to be extended and refined in order to better support the mapping tasks (presentation of the different offers to the end-user, mapping) of the converter.

The current description of the application profile is kept in one monolithic DTD. In fact the description of some general components (like the codecs, or the session characteristics) is described in the common DTD and may be implemented or valid for many applications. That means that there is redundant information in the concrete XML files (for one concrete application). One of the refinements consists in creating a session characteristic profile as well as a codec profile that will be referenced by the application profile in order to avoid redundancy, and redundant tests in case of same implemented codecs.

At the present a profile is a mix of general information and AQUILA related information. One of the next refinement consists in separating general information from AQUILA related information and, if possible, in finding rules governing the mapping between them.

The parsing of the profile to produce the GUI is not so effective as expected: complementary information is needed in the profile in order to optimise the GUI production.

3.3.1.4 Application profiles and complex internet services

3.3.1.4.1 QoS aware Complex Internet Service

Complex internet services are services offered by e.g. a content provider to a customer group in form of a web platform integrating, binding and presenting basic internet applications

For the second trial the MediaZine - a platform for music fans - will be implemented. As the current web technologies do not enable at application level a direct use of QoS technologies, there exist three possibilities to develop the MediaZine that are depicted and evaluated in the Table 3-1. The MediaZine will use the AQUILA QoS API in order to produce the QoS requests. It will integrate the user-friendly QoS selection/request (using the profiles of the integrated basic applications) in its overall GUI (see Figure 3-29).

Solution A	**Solution B**	**Solution C**
Bind basic internet applications together	Implement from null a service supporting QoS	Implement a service based on a QoS API
To implement:		

• Platform integrating the applications using proxies or API	• Basic applications • Platform • Signalling protocols	• Basic applications • Platform using API
Remarks		
Most efficient	Not efficient	Basic application to rewrite
AQUILA solution		prepared for future applica-tions

Table 3-1 Possible solutions to develop a QoS aware Complex Internet Service

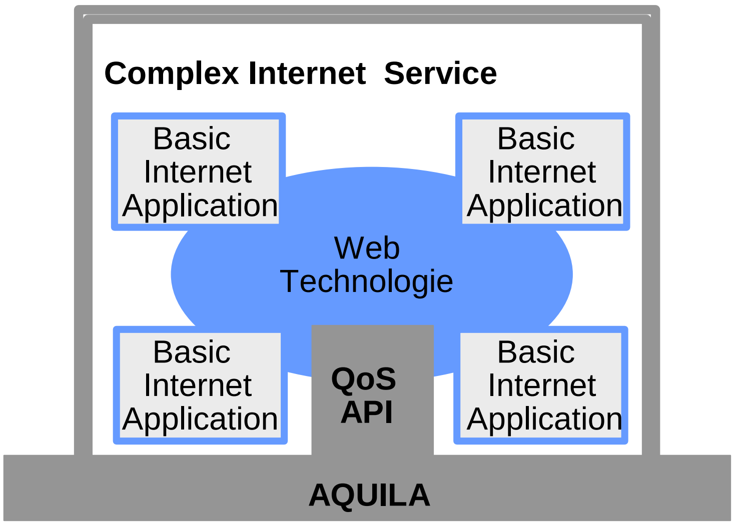
Complex Internet Service
Basic Internet Application
Basic Internet Application
Web Technologie
Basic Internet Application
QoS API
Basic Internet Application
AQUILA

Figure 3-29 Complex Internet Service and AQUILA

3.3.1.4.2 Overall architecture

The EAT API offers the access to end-user oriented application information by the way of the converter. The profile approach is used by the EAT API in order to provide to the “new” complex internet services an user friendly QoS interface towards the end-user.

The diagram below (Figure 3-30) gives an example of the utilisation of the application profile by the Complex Internet Service “MediaZine” (for more information on the EAT please refer to [D1201] and [D2201]). MediaZine will be implemented for the second trial.

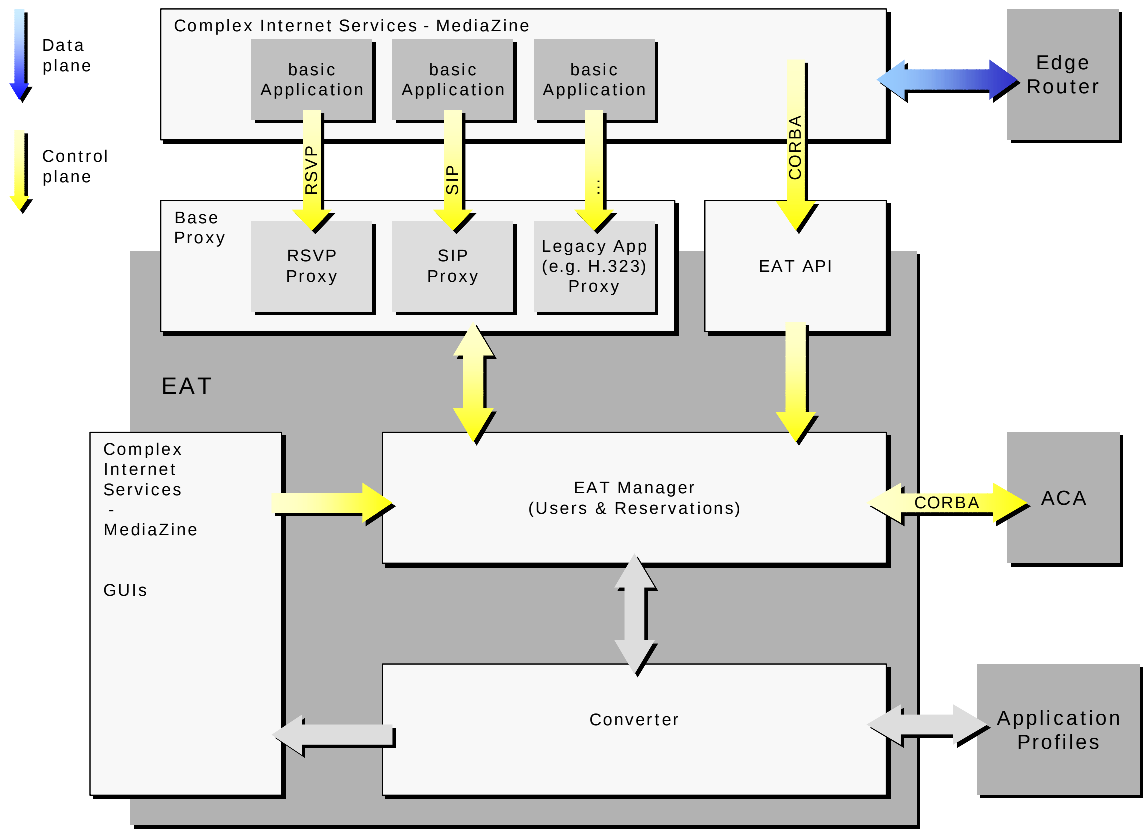

Figure 3-30 Block diagram representing the EAT

3.3.1.5 GUI

The GUIs correspond to the interface between the system and the end-user. The GUIs planned for normal end-users are based on the application profiles.

3.3.1.6 QoS monitoring

An important feature concerns the QoS monitoring. The end-user is paying for a “quality”. This quality is as mentioned in [ABOBA] very objective and depends on the kind of applications, their usage, the art of the agreement concluded with the ISP etc. Nevertheless a kind of monitoring can be important and can convince the end-user by for example monitoring the delivered bandwidth.

This feature can be supported by a measurement architecture that continuously measures the delivered bandwidth and gives feedback to the EAT that present it in a graphical form.

3.3.1.7 Reservation adaptation

Many legacy applications, to cope with the best effort QoS of the actual Internet adapt their resource needs to the resource level and implement QoS adaptation at application level. This means that the produced traffic is not constant and predictable. A solution associating packet sniffers and a measurement architecture could enhance the AQUILA architecture. In such a case the application profile structure has to be further refined.

3.3.1.8 Service presentation to the customer

Imagining the case of a SLA negotiation it is conceivable to base the negotiation on the predefined SLS. Two scenarios are thinkable: one presenting the SLS information in an advanced GUI for “professional user”, the other one presenting the SLS information in an user-friendly way. The latter assumes a mapping mechanism in order to translate the technical parameters into a user friendly “language”. An approach similar to the application profiles might be useful.

3.3.2 End-user application API

The Application Programming Interface (API) of the End-user Application Toolkit provides **application services/interfaces** for QoS. Via this API, application and service developers can directly access – i.e. by using a programming language – the QoS features of the AQUILA’s Resource Control Layer.

In detail, the EAT API provides services for:

- the end-user’s authentication against the network,
- the retrieval of by the RCL offered QoS network services (predefined SLS’),
- the retrieval of Service Level Agreements (SLAs),
- the request and the release of QoS reservations,
- as well as the retrieval of status information concerning QoS reservations (accounting, monitoring results, ...).

In contrast to signalling protocols, which serve applications at network level, the API serves applications at application level. The advantages of this approach are:

- The API is a single component to be used by all applications. In the case of the proxy dealing with signalling protocols, a separate proxy must exist for each supported signalling protocol. This causes a very high effort.
- Predefined SLSs (network services) and SLAs can be presented towards applications and end-users.

- With the API it is possible for an application to get feedback from QoS measurements and accounting information.
- The API can easily allow several requests per application session.
- End-users can directly be involved in choosing the QoS level.

With signalling protocols and proxies it is difficult if not impossible to realise the above mentioned features.

However there is one major disadvantage of the API approach:

- An application must be modified in order to access the API. This is not possible for many existing applications.

Application development is not the focus of the AQUILA project. Instead, we propose to create a new complex Internet service (e.g. Mediazine). Such a service encapsulates unmodified basic Internet applications and does the reservation requests for them. The realisation of such a service is much more efficient than to develop/modify basic Internet applications.

For the support of legacy, basic Internet applications, the Proxy approach is used.

Furthermore, there are two more or less contrary requirements for the API:

1. The API must provide the full functionality of the AQUILA approach in order to allow appropriate reservations. I.e. it must offer an interface that is strongly related to the reservation request interface of the ACA. It is therefore not generic but tailored for AQUILA purposes.

2. The API should also be as generic as possible in order to be widely accepted. It can be based on and/or can contribute to standardisation activities of the Internet community.

Therefore, we propose a twofold way: The EAT API, placed on the top of the EAT, is divided into two levels: the CORBA-based “internal” API and the Java-based “general” API (Figure 3-31).

The **Internal EAT API** summarises in a separate package all the interfaces of the EAT Manager towards applications and services which wants to directly access the EAT on an AQUILA-specific way. The graphical user interfaces of the EAT which are related to QoS reservations as well as the EAT’s scripting interface are already using it. The internal API is a proprietary interface designed for the specific purposes of AQUILA.

The **General QoS API** is a more generic one in order to provide QoS support not only based on the AQUILA’s RCL approach. The services that this API provides are more abstract and non-AQUILA-specific. Via the general API it should be possible to access different lower-level APIs such as the AQUILA’s internal EAT API, the RSVP API (RAPI), and the WinSock 2 API. For that, different “wrappers” are needed in order to map a general QoS request into a proprietary one.

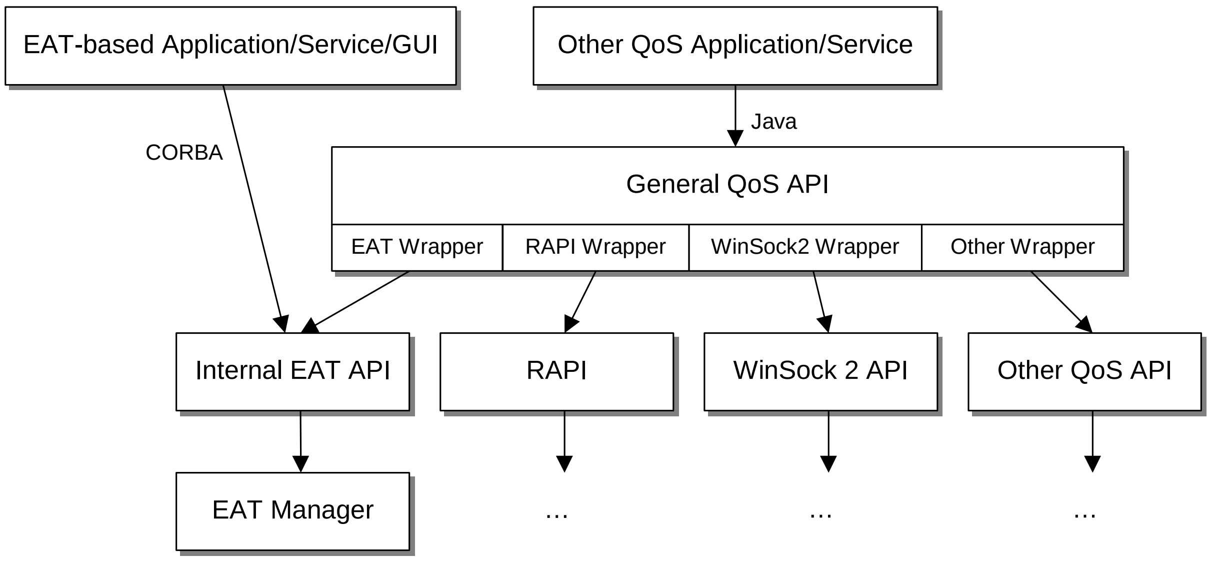

Figure 3-31: QoS APIs at different levels

The following two sub-chapters present the two APIs in detail. For the internal EAT API a first trial version already exists which is going to be revised for the second trial. This revision is made at design level. In contrast, the general QoS API has to be developed from scratch. Thus, its requirements have to be analysed first.

3.3.2.1 Internal EAT API

The Internal EAT API represents those CORBA interfaces of the EAT Manager – the central component of the EAT – which are related to end-user authentication, network services, QoS reservations, etc. The aim is that all EAT-based external applications, services, tools, and graphical user interfaces use this API to have access to the EAT Manager's features.

These features are based on the RCL's, more specifically the ACA's interface for user reservations. Consequently, the API provides similar interfaces for login, reservation request, reservation release, and logout. In contrast to the ACA, the reservation request, however, can be made in two modes:

- Advanced reservation mode for specialists: The request is made at the same level of abstraction as at the ACA, i.e. the full list of QoS, traffic, and flow parameters has to be specified. This mode is mainly for test and experimental purposes and for end-users/applications that are aware of the technical meaning of the request.

- Usual reservation mode for non-specialists: The request is made at a much higher level of abstraction. By using the pre-defined application profiles, the API provides abstract, well understandable QoS options to be selected by "normal" end-users. This mode should therefore be the *usual* way to request for QoS via the EAT.

The internal API, moreover, allows reservations for applications that dynamically negotiate data port numbers. In this case, the EAT Manager asks the Proxy to complete the reservation request “form”.

The above mentioned functionality of the API was already realised for the first trial. For the second one, some modifications are necessary due to the following reasons:

- The API must provide interfaces and methods for reservation groups. For that reason, on the one hand, the API provides four new methods for reservation group requests. On the other hand, two new interfaces represent the modified view to a reservation (session): *Single*QoSSession and *Grouped*QoSSession both inheriting from the former, now “abstract” QoSSession interface.
- The use of the three different interfaces for QoS requests is actually complicate and inflexible due to some CORBA restrictions. Instead, the QoSSessionRequest interface now includes the concerning methods for advanced and usual reservation requests.
- The same interface now foresees the retrieval of SLAs of an end-user.
- The new interface QoSRequester may be implemented by the client application, in order to be informed when something happens with the requested reservation. This mechanism is of importance mainly for so-called provisional reservations for which the Proxy detects the necessary flow data *after* requesting the reservation *and* then starting the application.

The new design model of the internal API considers the requirements above (Figure 3-32). The scenario for the usage is as follows:

1. Login is to allow end-user authentication. It also acts as a “factory” for a new QoSSessionRequest object when an end-user successfully logs in.
2. QoSSessionRequest is the basic handler for an end-user. It is to logout, to get information about the established SLAs, and to allow single and grouped reservation requests; both can be done in the advanced as well as in the usual mode. Also the QoSSessionRequest interface acts as a factory for a new QoSSession object when a reservation request succeeds.
3. Depending on the kind of the request, a new GroupedQoSSession or SingleQoSSession is created. Both specialise QoSSession, which is in general the representation of an active reservation. It allows the retrieval of accounting and QoS monitoring information, and the reservation release.

   3.1 A GroupedQoSSession consists of other (sub-) groups or (at the deepest level) of single sessions. A new member can join or leave this group. Last but not least, a GroupedQoSSession can be for a bi-directional reservation.
4. The aim of QoSRequester is to have a reference to the QoS requesting application.

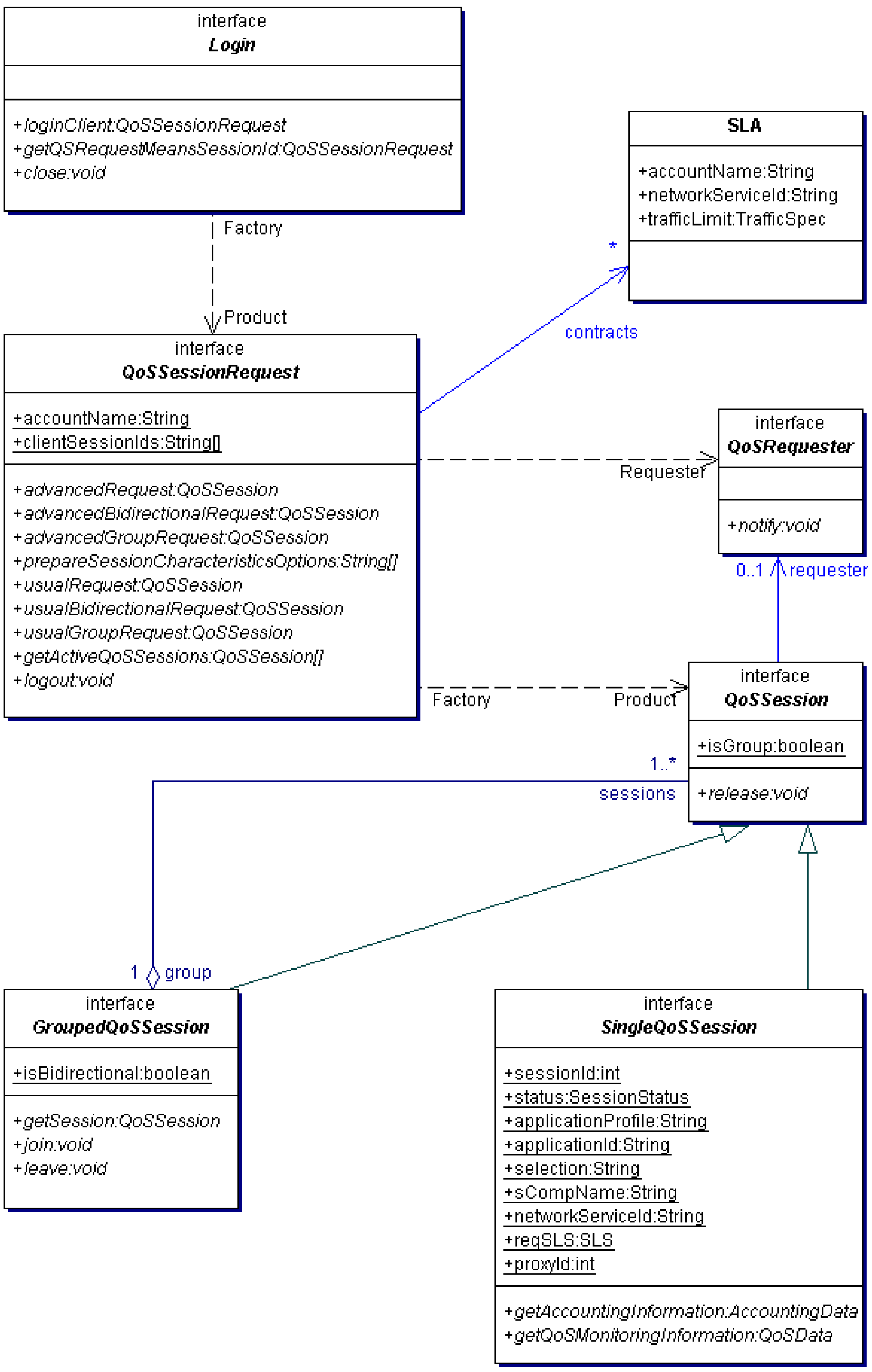

Figure 3-32: The Internal EAT API, part 1

As shown in the figure above, the content of a GroupedQoSSession may not only be single QoS sessions but could also be other groups. By using the composite pattern, the EAT API allows to build hierarchical group structures for multidimensional reservation groups. Such a group may consists, for example, of two dimensions like in a conferencing scenario: one for the calling partners, and one for the different service components such as “video” and “audio”.

However, this separation only exists within the EAT; for the ACA the hierarchical reservation group is mapped into a flat one.

Besides the above mentioned interfaces, two additional exist (Figure 3-33):

- ServiceDistributor shows the by the RCL provided network services, may be useful to negotiate a new SLA for an end-user.
- ApplicationManager shows the by the EAT provided application profiles (for usual requests), and the installed proxies (if needed by an application).

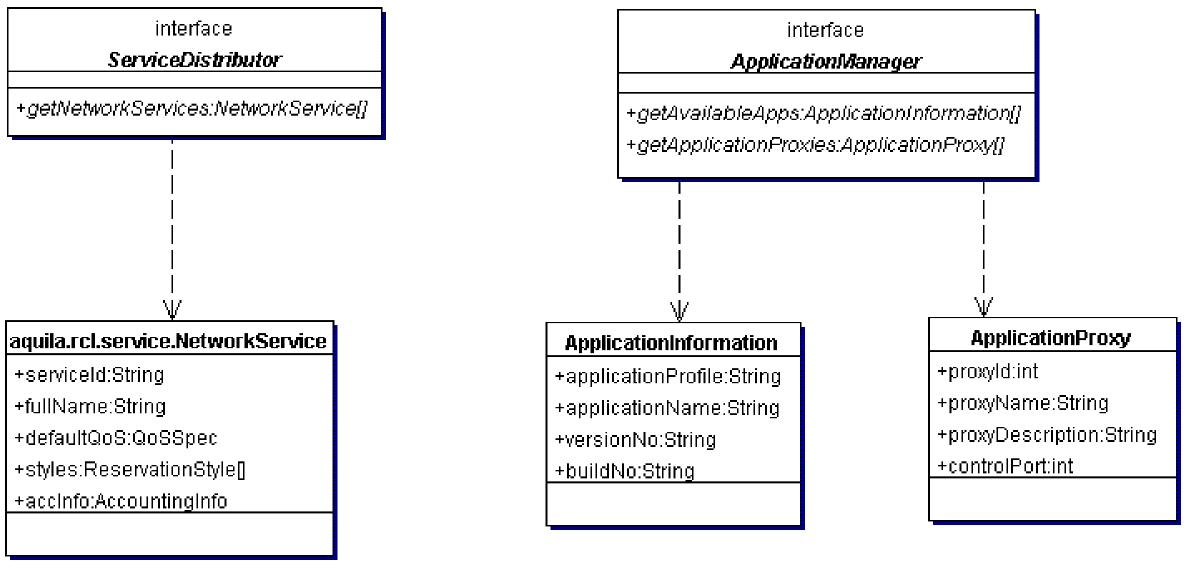

Figure 3-33: The Internal EAT API, part 2

The detailed specification of the Internal API can be found in chapter 6.3.

3.3.2.2 The General QoS API

3.3.2.2.1 Related work

First of all, the following short analysis of the related work concerning high-level QoS APIs gives an impression how such an API can be structured and what is still missing:

Java QoS API (JQoSAPI)

JQoSAPI is a platform independent API for QoS networking based on Java by the University of Ulm [JQoSAPI]. It provides *"an abstraction from QoS capable networks and allows a simple specification of QoS parameters"* [Kassler99]. The current release is based on ATM and AAL, and for Windows NT only. However, RSVP and other QoS approaches are not included yet.

OSS Quality of Service API

The OSS (Operations Support Systems) Through Java Initiative is going to define a Java QoS API [OSS-QoS] that offers a common way for applications to *manage* different kinds of Java-enabled QoS systems. For example, the control of SLAs will be provided by this API. The platform for the OSS QoS API will be the Java 2 Enterprise Edition. The project is still in the specification phase; no implementation is available yet.

Internet 2 QoS API proposal

A very similar approach to the here proposed General QoS API is the Internet 2 QoS API [I2-QoS-API]. It also provides an abstraction of some underlying existing APIs, such as RAPI and WinSock 2, and defines some basic QoS functions for authentication; retrieving QoS profiles for the different underlying protocols, e.g. RSVP; as well as binding, rebinding, and unbinding applications to QoS sessions. The API will be implemented in C++ and is therefore *not* platform-independent.

Quality of Service Management Environment (QoSME) & QoSockets

The QoSockets includes the:

"1. Support of single API for transport layer QoS negotiation, connection establishment, and data transmission; and of single API for OS QoS negotiation.

2. Support of a single QoS negotiation protocol.

3. *Generality across application QoS needs.*

4. *Automatic management of application QoS needs.*

QoSockets are available for Solaris and Linux and support RSVP, ATM adaptation, ST-II, TCP/UDP, and Unix native protocols." [QoSME]

Note that the QoSockets are in fact a lower level API like usual sockets. First implementations exist for Solaris and Linux.

QoSockets are the runtime component for the so-called **Quality (of Service) Assurance Language (QuAL)**. It is a proposal for QoS support at higher level providing another program-

ming language for the development of multimedia applications. A QuAL compiler then *"translates QuAL abstractions into calls to QuAL runtime components"* [QuAL].
To conclude: The above discussed APIs point out to the basic requirements of a general, high level QoS API:

"For such an interface to be useful, it must provide (at a minimum) the following capabilities:

- *Provide necessary credentials for* ***authentication*** *of QoS requests*
- *Be responsive to messages concerning environmental changes that impact the status of the* ***QoS service***
- *Generate* ***QoS parameters*** *that are acceptable to the {BB/Edge Router???}*
- *Easily* ***map*** *to underlying API's, such as WinSock 2 and RAPI as well as others*
- *For the HOST API to be usable, it must not only allow advanced developers to tweak each QoS 'bit' but also provide a* ***simple interface*** *for the developer. Such an API would provide:*
    - *the ability to choose a* ***QoS profile*** *based on easily understood attributes (video, audio, etc.)*
    - *the ability to access a 'profile service' to obtain a predefined profiles describing QoS attributes*
    - *the opportunity to leverage existing domain security infrastructure for authorisation of QoS requests*
    - *a mechanism to* ***negotiate*** *for available bandwidth if the QoS request cannot be satisfied*
    - ***feedback*** *on the business attributes of the QoS service provided (cost, window of available service, etc.)"* [I2-QoS-API]

In addition, it can be mentioned, that such an API should provide an uniform**, platform-independent** way to use it.

3.3.2.2.2 Proposal

The following proposal considers both: the experiences of the AQUILA approach as well as the above mentioned requirements, which are in fact very similar.

To start with an analysis, the possible uses cases have to be taken into account. Two views have to be distinguished: The (human) end-user/customer's view of the (QoS-aware) application (Figure 3-34), and the application's view of the QoS API (Figure 3-35).

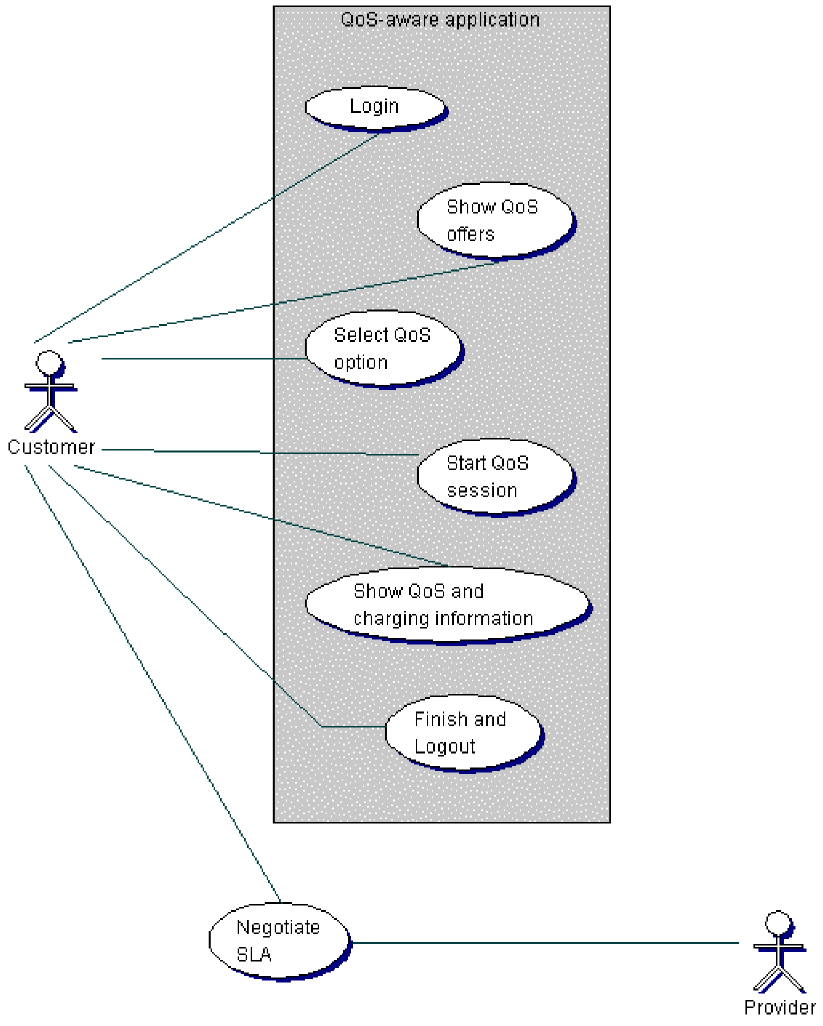

Figure 3-34: Use cases of an QoS-aware application for an customer

As shown above, interaction between the customer and the QoS-aware application is necessary: The QoS-aware application must support user identification, and may offer its services/media with optional QoS support. The customer has to have the possibility to choose between QoS options, because he/she has the SLA with the QoS provider, and he/she will pay for them.

For the realisation of the scenarios above the QoS-aware application makes use of the QoS API as depicted in the following figure:

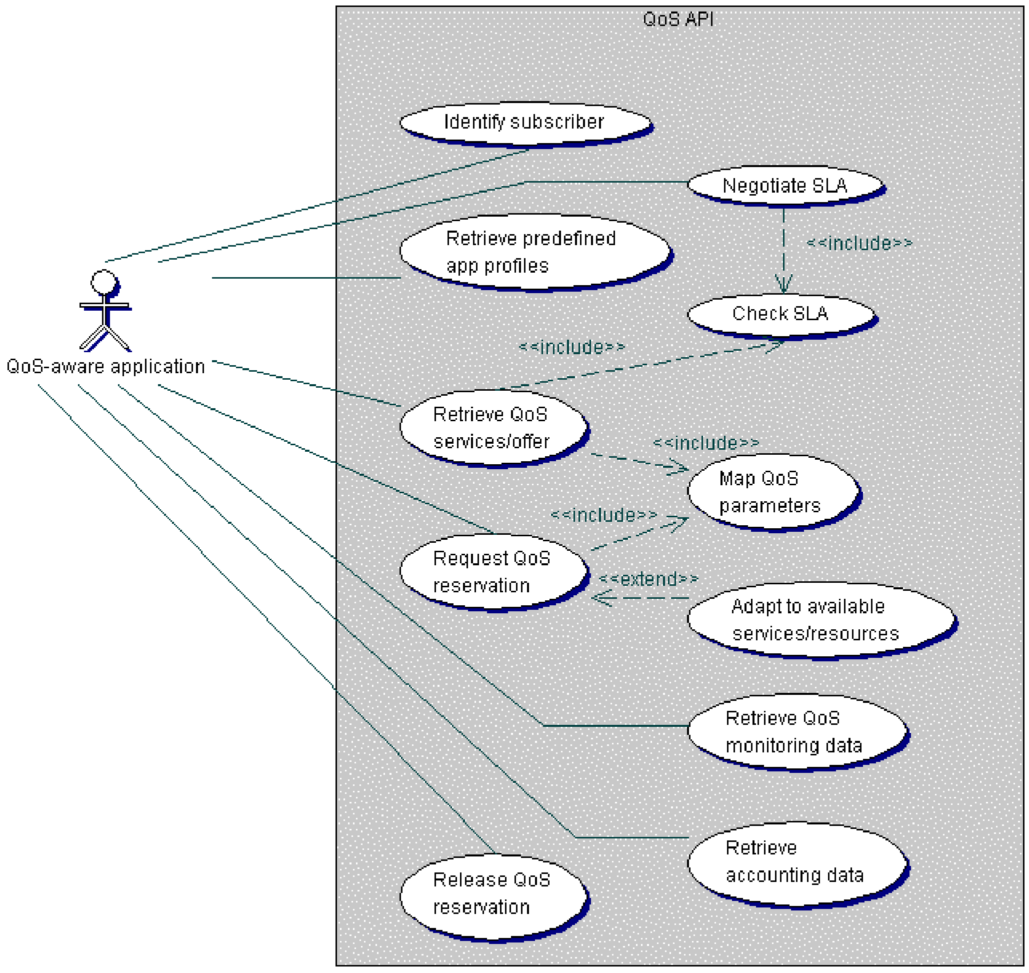

Figure 3-35: Use cases of the QoS API for an application

The following analysis model relies on the uses cases above and depicts the most important classes that the General QoS API must provide (Figure 3-36).

Remarks: The most of the classes, attributes, methods, and labels are self-explanatory. However, there might be some questions to be clarified:

- Authenticator will probably a singleton class.
- A Subscriber indirectly request for a reservation (QoS Session) by selecting a predefined QoS Option which is provided by an Application Profile.
- The SLAs of a Subscriber may restrict the number of available QoS Options.
- The available QoS Options also depends on the usage of a concrete, underlying QoS Protocol (QoS technology) such as AQUILA, RSVP, etc. The established reservation (QoS Session) relies on this “protocol”, too.
- A Multi QoS Session (i.e. a reservation group) consists of other Multi and/or – at the deepest level – Single QoS Sessions.

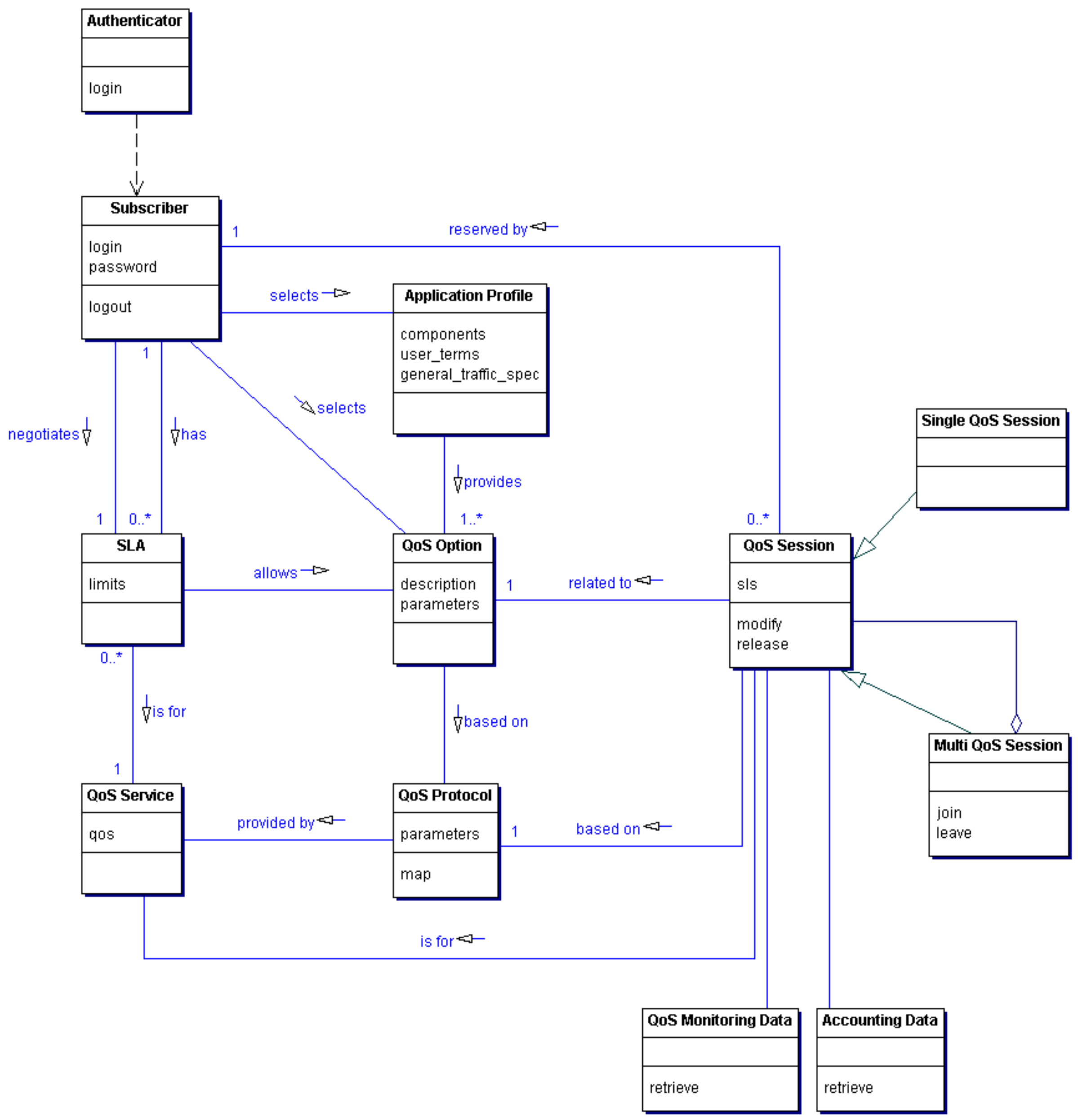

Figure 3-36: General QoS API analysis model

3.3.3 Protocol gateways (SIP)

3.3.3.1 Introduction

One of the important objectives of the AQUILA architecture is to transparently support legacy applications. In general, legacy applications cannot be modified in order to use a particular QoS API. Moreover, the parameters needed for a QoS request for such an application (e.g.

TCP/UDP port numbers, traffic profile) cannot be examined or even influenced. Therefore, the EAT should use a variety of mechanisms in order to detect or measure those parameters.

In the AQUILA architecture, a reservation request from the EAT to the ACA comprises various parameters: source and destination addresses and ports of the IP flow, its traffic profile in the form of a TSpec as well as its QoS requirements (selected by the user only in the second trial). In the case of legacy applications, not all parameters are known to the user, so that a manual reservation can be made. To this end, the EAT provides several mechanisms that provide us with those parameters. Application Profiles have been defined, that capture the QoS needs of a particular application, by providing the necessary set of parameters for a QoS request. Protocol Gateways or Proxies are used for the detection of TCP/UDP port numbers of application flows.

Many legacy applications open data connections using well-known port numbers, or ports that can be pre-configured to a known value. However, other applications communicate with dynamically negotiated port numbers. Such applications are mainly IP telephony or videoconference services that use connection establishment protocols such as SIP or H.323. Nevertheless, the same concept applies also to other applications that communicate through standard connection protocols (e.g. streaming applications that use RTSP). In order to make a resource reservation for such an application, the port numbers of each of the application flows should be known at the time of the QoS request. However, those parameters are negotiated between the communication entities during the connection set-up. Moreover, there is often no way for the user to manually select a particular port number.

What is needed, is the interception of connection set-up messages and the detection of those parameters. Then they will be passed over to the EAT for the completion of a QoS request. The EAT uses the entities called Protocol Gateways or Proxies in order to get those parameters.

3.3.3.2 Protocol Gateways in general

3.3.3.2.1 Functionality

The protocol gateways used in AQUILA act as Application-level Proxies. The Application-level Proxy resides between the application and the network. In general, such a proxy interferes with the connections between two hosts. The client host, instead of connecting directly to the server, contacts the proxy. The proxy, in turn, connects with the server and relays packets coming from the client, to the server and vice versa. In this way, the port numbers of the Proxy – server connection are known to the Proxy, who is able then to make a QoS reservation for the IP packets of this connection.

In AQUILA, Protocol Gateways do not just participate in the signalling message exchange between client and server, but also they interpret the exchanged packets. By intercepting and interpreting the exchanged messages, the protocol gateways can find all the important information pertaining to a reservation. In the general case, this information is the source and destination addresses and ports of the flows that carry user data. In special cases, as we will see next, this information may be broader, covering also information about the traffic profile of those flows.

For a Proxy to operate properly, the communicating entities must have been configured to use one, so that all connection set-up passes through it. If there is no such option, the signalling messages should be intercepted by a filter and forwarded to the appropriate proxy.

3.3.3.2.2 Operation

In the general scenario of operation followed during the first trial, the EAT Manager asks the Proxy to be alert for the launching of a new application. This is performed by instructing the Proxy to expect a new flow of a specific service component (audio, video, or data) from a specific host. The Proxy Manager forwards this request to the appropriate Protocol Gateway, according to the instructions of the EAT Manager. When the application initiates a new connection, it will go through the Proxy. After the particular proxy has translated the connection establishment messages and detected the necessary addresses and port numbers, it feeds them to the Proxy Manager, who in turn, forwards this information to the EAT Manager.

The Protocol Gateways should also be able to operate on their own. When they detect the establishment of a new connection they should automatically notify the EAT. The EAT then, will notify the end-user and ask for permission to make a reservation for this session.

However, this process may be prolonged and the connection will have started for quite a long time until the user is notified for his approval. Moreover, due to the web-based architecture of the EAT GUIs, it may be difficult even to find a way to asynchronously notify the user. Therefore, in order to simplify this process, the EAT may store some user preferences that allow it to set automatically reservations for QoS-needy traffic, especially IP telephony, with prearranged traffic parameters.

3.3.3.3 Case study: SIP Proxy

3.3.3.3.1 Overview of SIP

The Session Initiation Protocol (SIP) is a signalling protocol for Internet conferencing, telephony, presence, event notification and instant messaging. The protocol initiates call set-up, routing, authentication and other feature messages to endpoints within an IP domain. SIP is independent of the packet layer and only requires an unreliable datagram service, as it provides its own reliability mechanism. While SIP typically is used over UDP or TCP, it could, without technical changes, be run over IPX, frame relay, ATM AAL5 or X.25. SIP is a text-based protocol, resembling the hypertext transfer protocol (HTTP) and simple mail transfer protocol (SMTP). SIP uses Session Description Protocol (SDP) for media description.

SIP follows the client/server architecture. The main entities in SIP are the User Agent, the SIP Proxy Server and the SIP Redirect Server. The User Agents, or SIP endpoints, function as clients (UACs) when initiating requests and as servers (UASs) when responding to requests.

User Agents communicate with other User Agents directly or via an intermediate server. The User Agent also stores and manages call states.

SIP intermediate servers operate either as proxy or as redirect servers. SIP Proxy Servers forward call signalling from the User Agent to the next SIP server or User Agent within the network. SIP Redirect Servers respond to client requests and inform them of the requested server's address. In this way, User Agents may connect directly to the desired User Agent. To maintain scalability, the SIP servers can either maintain state information or forward requests in a stateless fashion.

3.3.3.3.2 Requirements for a QoS SIP Proxy

For the purposes of the AQUILA project, a SIP Protocol Gateway will be used that implements a subset of the functionality of a SIP Proxy server. The main tasks of a SIP Proxy (or Redirect) Server are:

- to find the location of a user (to find the specific workstation/PC that the called user is currently logged on)
- to pass the resulting connection through a firewall.

More than one proxy may lie in the signalling path from the caller to the called party. A general scenario is depicted in the following figure:

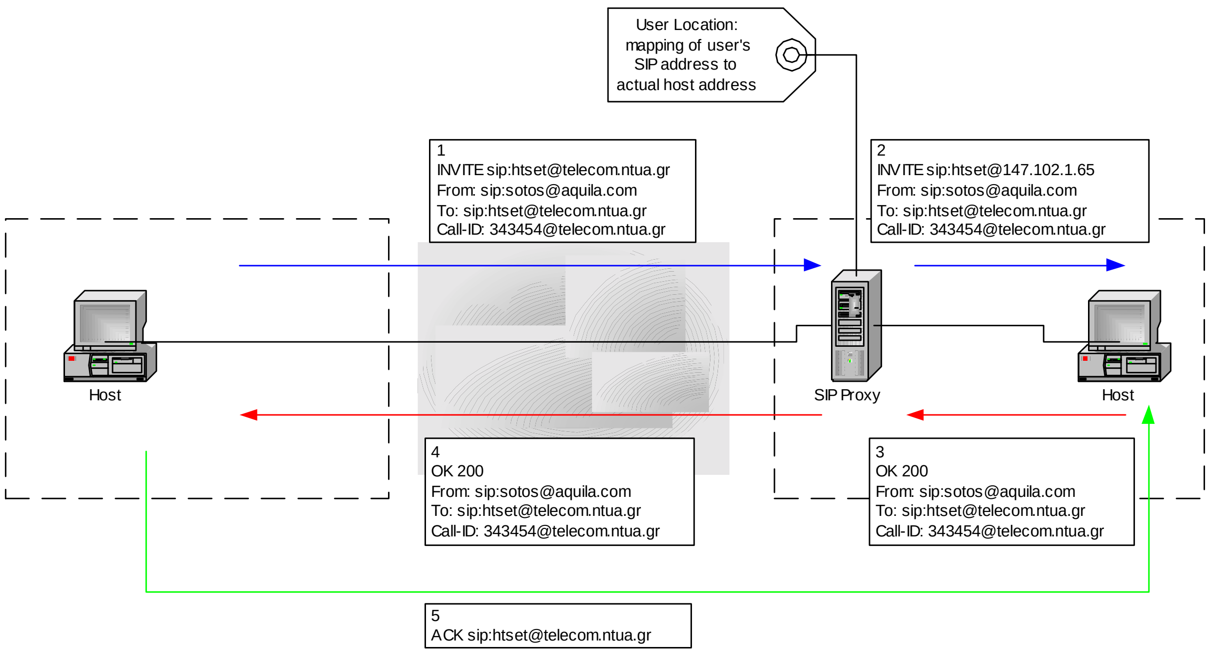
User Location: mapping of user's SIP address to actual host address
1
INVITE sip:htset@telecom.ntua.gr
From: sip:sotos@aquila.com
To: sip:htset@telecom.ntua.gr
Call-ID: 343454@telecom.ntua.gr
2
INVITE sip:htset@147.102.1.65
From: sip:sotos@aquila.com
To: sip:htset@telecom.ntua.gr
Call-ID: 343454@telecom.ntua.gr
Host
SIP Proxy
Host
4
OK 200
From: sip:sotos@aquila.com
To: sip:htset@telecom.ntua.gr
Call-ID: 343454@telecom.ntua.gr
3
OK 200
From: sip:sotos@aquila.com
To: sip:htset@telecom.ntua.gr
Call-ID: 343454@telecom.ntua.gr
5
ACK sip:htset@telecom.ntua.gr

Figure 3-37: SIP scenario

The required information for a QoS request includes the port numbers of the audio/video connection(s). Those parameters are negotiated between client and server User Agents during the set-up of the connection. An example exchange of SIP messages for a simple audio call is presented in the following (taken from [RFC2543]):

```
   C->S: INVITE sip:watson@boston.bell-tel.com SIP/2.0
         Via: SIP/2.0/UDP kton.bell-tel.com
         From: A. Bell <sip:a.g.bell@bell-tel.com>
         To: T. Watson <sip:watson@bell-tel.com>
         Call-ID: 3298420296@kton.bell-tel.com
         CSeq: 1 INVITE
         Subject: Mr. Watson, come here.
         Content-Type: application/sdp
         Content-Length: ...

         v=0
         o=bell 53655765 2353687637 IN IP4 128.3.4.5
         s=Mr. Watson, come here.
         c=IN IP4 kton.bell-tel.com
         m=audio 3456 RTP/AVP 0

… skipped some messages …

   S->C: SIP/2.0 200 OK
         Via: SIP/2.0/UDP kton.bell-tel.com
         From: A. Bell <sip:a.g.bell@bell-tel.com>
         To: <sip:watson@bell-tel.com> ;tag=37462311
         Call-ID: 3298420296@kton.bell-tel.com
         CSeq: 1 INVITE
         Contact: sip:watson@boston.bell-tel.com
         Content-Type: application/sdp
         Content-Length: ...

         v=0
         o=watson 4858949 4858949 IN IP4 192.1.2.3
         s=I'm on my way
         c=IN IP4 boston.bell-tel.com
         m=audio 5004 RTP/AVP 0
```

In this example, the calling party requests an audio connection (INVITE message), stating that he will receive on port 3456 and proposes to use the PCMU (ITU-T u-law) codec during the call. In its final message (200 OK), the called party does not pose any objection to the port number, and says that it will use port 5004 to receive its own data.

We intend to use a SIP Proxy, located in the caller's premises that will intercept the messages and find those port numbers. The Proxy may also inform the EAT about the codecs used during the session, so that a correct reservation may be requested. The above scenario is depicted in the following figure:

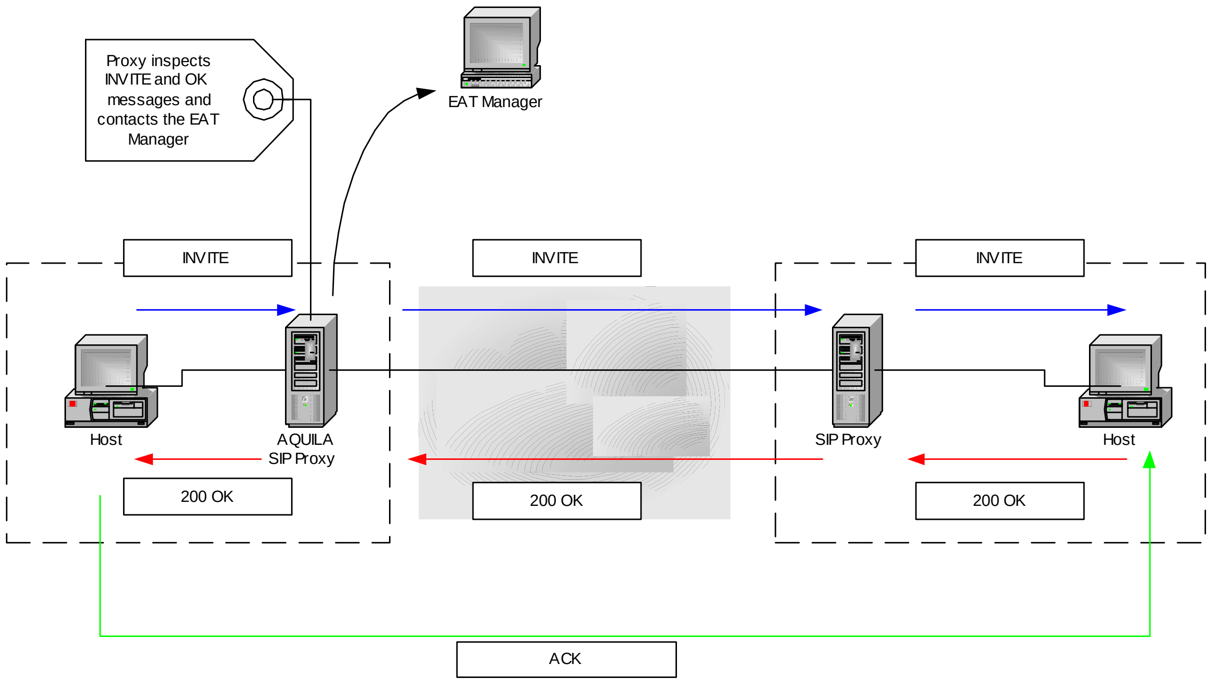

Figure 3-38: QoS SIP scenario

For the specification of the QoS SIP Proxy, as it will be called, full compatibility with the current SIP Proxies used in the market today will be pursued. As a matter of fact, the QoS SIP Proxy should be specified as an add-on to current SIP Proxies and the intricacies of real proxies should be kept in mind.

However, for reasons of simplicity, some assumptions will be made. The Proxy will not have the full functionality of a SIP Proxy, especially as far as User Location services or firewall traversal are concerned (out of the focus of AQUILA). Also, during the trials, simple naming schemes will be used. A user should be phoned directly (by using his IP address: e.g. htset @ 147.102.1.7) instead of using his SIP or e-mail address (e.g. htset@telecom.ntua.gr). Finally, the SIP Proxy will not interfere in the SIP message exchange, unless it is needed. For example, it will not “fork” an invitation, but it should alter the message headers (e.g. the “via” header) so that the backward flow of messages will pass through it.

3.3.3.3.3 SIP Proxy Operation

3.3.3.3.3.1RFC 2543

The objective of the integration of SIP and resource reservation is to *ensure* that an IP call session will receive the requested resources, before the call is started, that is, before the called party answers the call. In other words, the resource reservation procedures must have completed successfully (or not), before a 200 OK method is sent back from the called to the calling party. There are two kinds of sessions: **QoS-Assured** and **QoS-Enabled**. An QoS-Assured session will complete only if the resources are available and allocated. If not, the network will not complete the call. In a QoS-Enabled session, the resources may or may not be allocated. In any way, the session will be completed and the call will start.

The current specification of SIP in RFC 2543 does not foresee provisional responses, in a way that the call initiation could be partially completed, awaiting the reservation procedure to respond. Moreover, the SDP information from the called party is included only in the 200 OK responses. This method is sent immediately *after* the called party answers the phone. As a result, the session will start without the knowledge if the reservation was honoured or not. The exchange of messages in this case was shown in the previous figure.

A QoS SIP Proxy would, in this case, provide only QoS-Enabled sessions. This is acceptable in the context of the AQUILA architecture, since there is no official extension up to now that has reached RFC status. This Proxy will operate and provide the port number information in two ways:

- on demand from the EAT Manager
- upon detection of a new SIP connection, according to installed preferences

3.3.3.3.3.2Extensions to SIP for QoS support

In order to explicitly support QoS for IP telephony, a number of Internet Drafts have emerged lately. Most of those drafts have been replaced by draft-ietf-sip-manyfolks-resource-01 [SIPRES]. In this draft, it is suggested that connection establishment and resource reservation should in general be de-coupled. That is, SIP should not be used also for resource reservation, but a single QoS mechanism should be used for all applications. However, a total de-coupling is not feasible, since vital information that is used for resource reservation (addresses, ports, codecs) is not known in advance, but is dynamically allocated.

This draft specifies some extensions to SIP and SDP. A new message is added (COMET method), used to confirm the successful reservation of resources and allows the completion of a connection. Also, two new SDP attributes are defined: “qos” and “security”. The qos attribute indicates whether end-to-end resource reservation is optional or mandatory and in which direction (send, recv, sendrecv). An example SDP description, carried in an INVITE message is the following. In bold fonts are the proposed additions:

```
v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
m=audio 49170 RTP/AVP 0
a=qos:mandatory recv confirm
m=video 51372 RTP/AVP 31
a=secure:mandatory sendrecv
m=application 32416 udp wb
a=orient:portrait
a=qos:optional sendrecv
a=secure:optional sendrecv
```

In the draft, it is proposed that after both User Agents have agreed on the parameters of the connection (ports and codecs) they engage in the reservation procedure. In contrast to the SIP specification, the called party responds to the INVITE method with a 183 method that also contains an SDP description of the audio stream the called party is about to send. At this point, a provisional acknowledgement is exchanged and reservation is about to start.

In draft-ietf-sip-manyfolks-resource-01 [SIPRES], the chosen reservation mechanism is the RSVP protocol. After the exchange of provisional acknowledgement is completed, the hosts start exchanging RSVP messages. Only after resources have been definitely reserved, may the SIP connection be completed: the called party's phone will ring and upon his/her answer, the voice packet exchange will start. In the following figure an example exchange of packets is illustrated.

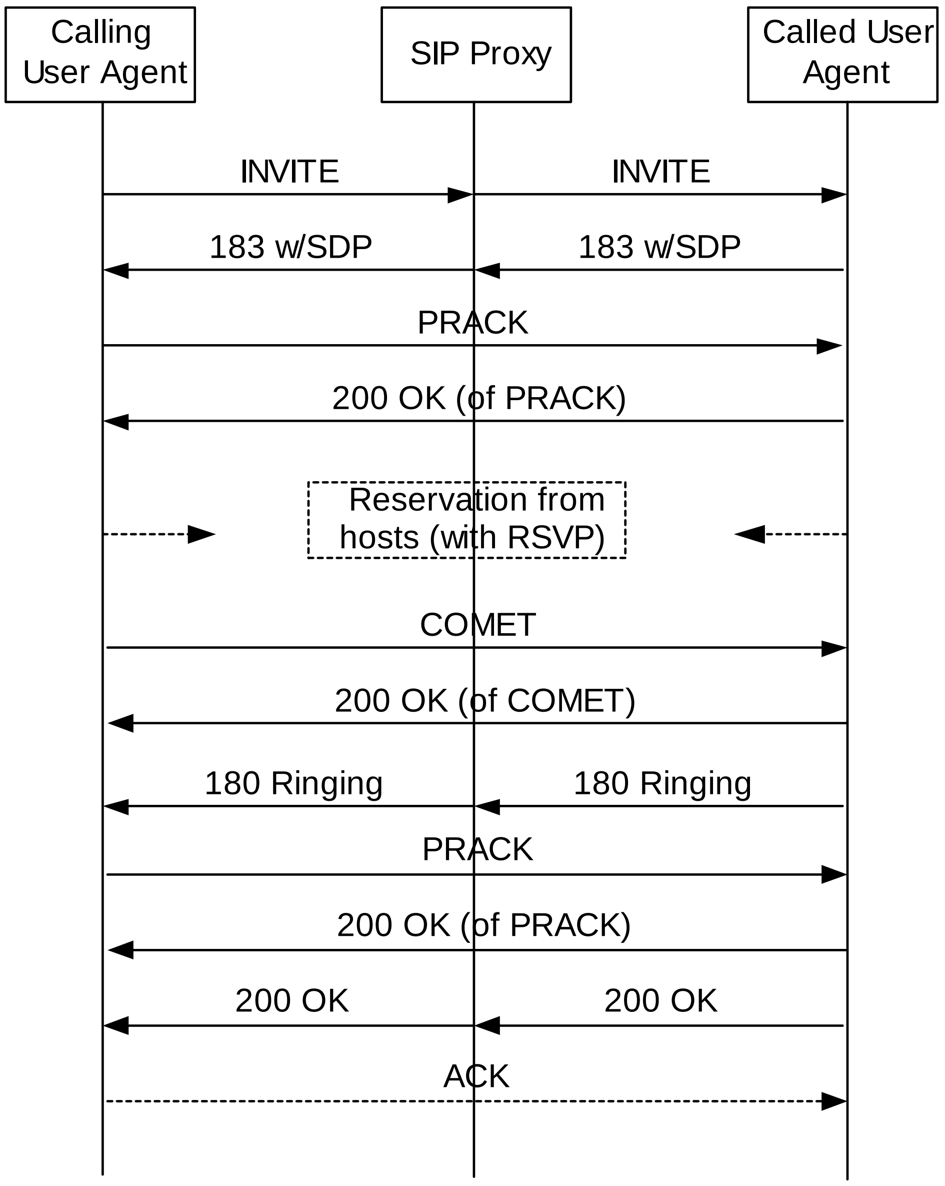

Figure 3-39: Sample message flow

The proposed AQUILA Proxy solution seems to be aligned with the approach envisioned for the support of resource management for SIP-initiated calls. In AQUILA, the Proxy will take part in this message exchange and will invoke the resource reservation mechanisms of AQUILA on behalf of the hosts. The definition of the new SIP messages and SDP attributes proposed in draft-ietf-sip-manyfolks-resource-01 [SIPRES] will enable the Proxy to provide QoS-Assured sessions, in addition to the QoS-Enabled ones.

The approach depicted in draft-ietf-sip-manyfolks-resource-01 has some limitations, in comparison to the proxy approach. First of all, the hosts should support the RSVP protocol (or another QoS signalling protocol) if they are to make QoS requests. It is questionable whether simple SIP phones will be capable of running also an RSVP daemon. Moreover, non-RSVP aware (and non-QoS aware in general) hosts will not be able to leverage the network's QoS functionality. Finally, this solution ties SIP closely with RSVP, even though it is not certain that RSVP will be the actual QoS mechanism in the future networks, given its scalability limitations.

On the contrary, in the proxy approach, QoS functionality is present only in the network and is not required by the end-hosts. Moreover, it can be used with a variety of reservation mechanisms (AQUILA, Bandwidth Brokers in general). However, if this approach is accepted within the IETF as an RFC, the effect of this standard on the EAT architecture should be investigated.

3.3.3.4 Conclusions

As mentioned, the current specification of SIP can be complemented with a QoS SIP Proxy to provide QoS support. Although the produced reservations will not be assured before the called party answers a call, this solution is still considered very important for AQUILA.

The draft draft-ietf-sip-manyfolks-resource-01 will likely be accepted as an RFC, taking also into account the fact that is backed by many of the SIP driving forces in the Internet Community. It is important that the devised solution takes into account the proposed extensions so that the QoS SIP Proxy is easily updated in the future.

3.4 Management (QMTool)

QoS Management Tool is intended to provide the Administrator with a simple, user-friendly tool for the design of the resource pool tree, the overall configuration of the network components, the monitoring of the required parameters of the network elements and the notification of a network element failure.

It is important that it remains a simple tool, with no extra features that will deteriorate its performance and with as little interaction as possible with the RCL elements in order to produce a minimal network load. Particularly the monitoring functionality should strictly stick to the minimal required so that no complexity is introduced within the QMTool and the RCL components.

3.4.1 Functionality

3.4.1.1 Design of Resource Pool tree

The QMTool, at its current state, provides a simple interface for the design of the resource pool tree. A toolbar with a set of required buttons enables the design of the tree while imposing certain restrictions (a leaf is connected to only one resource pool) and facilitating the design (i.e. creation, deletion of resource pools).

Moreover the tree should be enhanced with the visualisation of EATs as well. This enhancement will only serve the failure detection of EATs (explained later).

A more detailed description of the design functionality of QMTool can be found at the "QMTool User Manual" (21ntu1001b.doc).

3.4.1.2 Overall network configuration

3.4.1.2.1 Resource Pools

QMTool should enable the initial configuration of all network components, namely the configuration of all elements of the Resource Control Layer that interact with the LDAP database during the start-up procedure of the network. These components include the Resource Pools and the ACAs (not the EATs), as the latter need to retrieve their configuration information from the LDAP database.

This initial configuration is enabled with the help of a graphical interface (dialog boxes) that provides a set of parameters (for the Resource Pools and ACAs) to be filled in with their corresponding values. These parameters define the Resource Share information for each resource pool (for all Traffic Classes). QMTool stores that information into the LDAP database in order to be retrieved subsequently from the Resource Pools and ACAs (current state of QMTool).

Moreover, a control mechanism could be made available in case of a faulty configuration of the resource pools. For example, QMTool should not permit the configuration of a father resource pool with fewer resources than its child.

3.4.1.2.2 Services

Additionally, QMTool should enable the creation of Services, namely Network Services and their corresponding Traffic Classes. Similarly, a graphical interface should be provided to the administrator so that the latter is able to define the QoS specification of the provided services and their corresponding traffic description. In the same way, that information is stored into the LDAP database in order to be available for use by the network elements.

3.4.1.2.3 Subscribers

Furthermore, the registration of users and of all their subscription information should also be a task of this tool. The administrator should be able to add subscribers for the Aquila network and define their authentication information as well as the services they have subscribed for.

To be noted that for the implementation of the aforementioned tasks of QMTool, the latter should only interface the LDAP database and no interaction with the RCL components will be required.

3.4.1.3 Monitoring of network elements

3.4.1.3.1 Monitored parameters

QMTool should provide a mechanism for the collection and monitoring of the necessary parameters of network elements. This will include for example the utilisation of traffic classes (reserved and total resources) and the general performance of network components. This monitoring functionality should aim at the evaluation of the resource pool algorithms and their efficiency at accommodating reservation requests.

For the collection of data, QMTool should interface the RCL components via CORBA. In particular, it should interact with the resource pools (i.e. their Resource Shares) in order to retrieve their current state (reserved and total resources) and monitor it in the required manner (according to the type of monitoring selected).

Any further interaction with lower level network components (DiffServ layer) should not be performed, as it would add more complexity to the tool and perhaps more network load.

3.4.1.3.2 Type of monitoring

Moreover, QMTool should support two types of monitoring. The simplest one should provide a mechanism for the instant triggering of predefined parameters of the resource pools. In this way, the administrator can have an instant view of the values of resource pool parameters when required. This type of monitoring could possibly serve the evaluation of the resource pool algorithm at the time the reservations are made.

The second type of monitoring should be a continuous one. Therefore, the administrator should be able to determine the parameter to be monitored and the period between two value retrievals. These values will be kept at a log file, which will be used for the graphical representation of the retrieved information. This type of monitoring aims at an overall evaluation of the network performance and of specific parameters of the network elements.

3.4.1.4 Failure Detection

QMTool should enable the detection of a simple failure scenario. This will only include the case where the RCA, the ACAs or the EATs have stopped running for any reason. The administrator, at any time, will be able to check the servers, namely the RCA, the ACAs and the EATs. To that end, it should interface via CORBA with the aforementioned servers in order to detect their state (whether they are up or down). This failure detection can be performed for each server separately or for all the servers at the same time.

4 Studies on related topics

4.1 MPLS networks

The AQUILA solution does not rely on MPLS to work, see Figure 4- case a). As several operators are implementing MPLS for their IP backbone, it is worth considering how the AQUILA solution is related to MPLS, describing a typical possible scenario, see Figure 4- case b). MPLS and AQUILA can coexist, but they do not necessarily need to have a strict interaction. Therefore the design of AQUILA architecture for the second trial does not need to deeply include MPLS aspects.

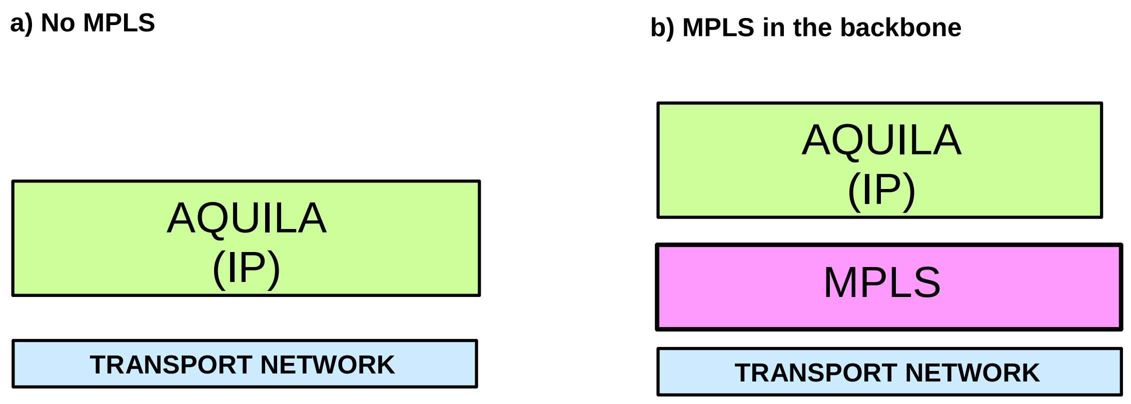

Figure 4-1: AQUILA and MPLS

The MPLS is assumed in the backbone and not in the access networks. Only Static LSPs are considered. The LSP are set-up manually by the operators for traffic engineering and to support VPNs. When traffic enters to an “MPLS zone” MPLS headers are placed in top of IP headers. Typically this is done in the PE (provider edge) device, see Figure 4- (the alternative solution would be to do it in Customer Edge – CE devices).

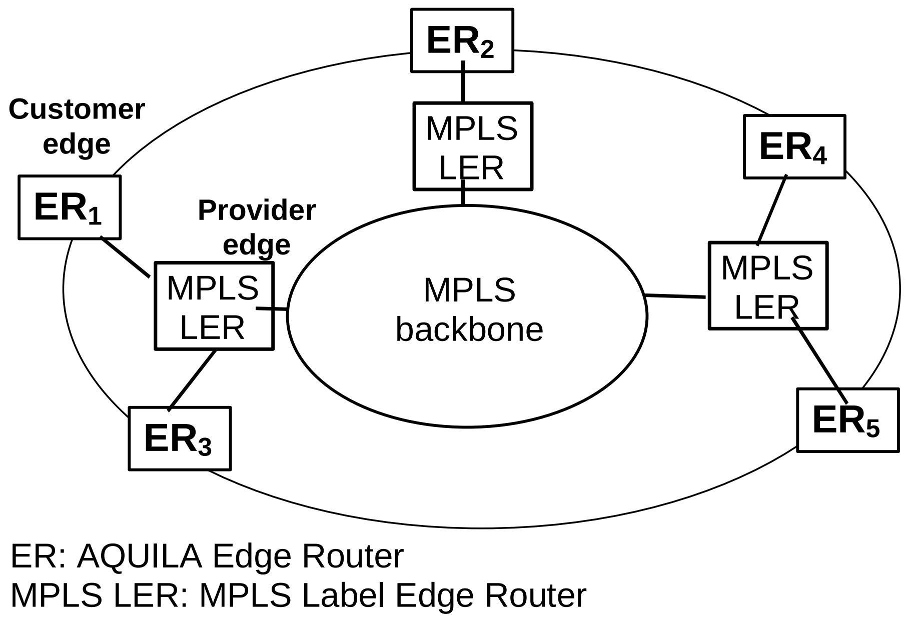

Figure 4-2: MPLS in the internal part

The interaction between AQUILA and MPLS is basically a DiffServ / MPLS interaction problem. The fundamental problem is how to map the QoS that has to be delivered to DiffServ classes into the MPLS world. The MPLS supports two models: L-LSP and E-LSP. A practical problem is the availability of these mechanisms in the MPLS routers. E.g. currently only E-LSP is implemented by Cisco. With E-LSP the EXP bits in the MPLS shim header are used to select one queue (out of a maximum of 8 different ones) in the outgoing interface of the router. Whatever mechanism is used, we assume that everything is handled by the operator at the MPLS level, i.e. the DiffServ (AQUILA in our case) has no visibility and no control on it. Therefore we can say that MPLS in transparent from the AQUILA point of view.

Let us consider now the VPN service. Note that the typical VPN service does not imply a guaranteed bit rate nor a specified QoS, but it is more a security and private routing service. A special routing table is used to forward the packets, so that independent routing space between different VPNs is provided. The LSP is selected according to the VPN information and it does not need to have a bandwidth nor a QoS assigned. If one wants to assign a bandwidth to a VPN, this is accomplished by means of MPLS traffic engineering. In this simple scenario there is absolutely no interaction between AQUILA and VPNs. This is represented in Figure 4-, where the VPN service and AQUILA are just like independent clients of the MPLS transport

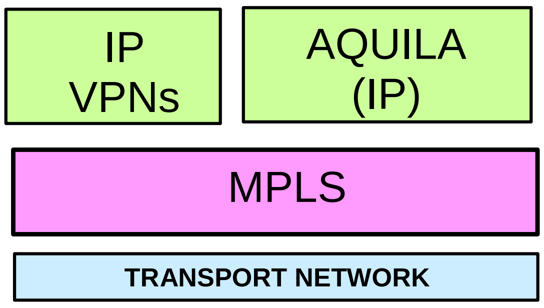

Figure 4-3: AQUILA, MPLS and VPN

4.1.1 Advanced issues

The basic co-existence scenario between AQUILA, MPLS and VPNs described above could be enhanced. These enhancements are out of the scope of AQUILA specification and implementation work and are shortly discussed hereafter as hints for future work.

There are two main aspects that can be enhanced: the interaction between AQUILA and MPLS for Traffic Engineering aspects, see (1)-Figure 4-, and the interaction between VPN and AQUILA in case VPNs should also provide/use QoS, see (2), Figure 4-.

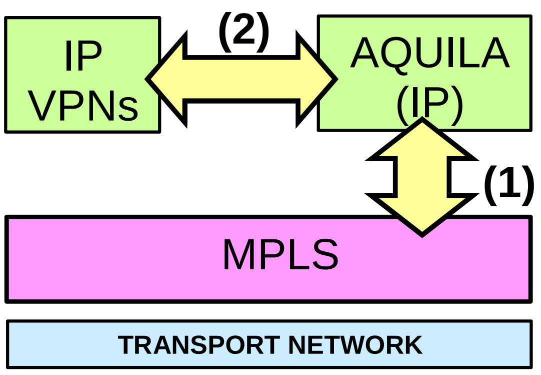

Figure 4-4: AQUILA, MPLS and VPN: enhanced interactions

AQUILA will not specify how to provide QoS and Traffic Engineering in the MPLS backbone. Note that even if a fully integrated solution would be theoretically optimal, it could be practically too complex… Some issue could be investigated at theoretical level. For example the definition of Traffic handling at packet level in MPLS core router could be investigated. In fact we assume that the MPLS backbone is able to provide a suitable transport to AQUILA Traffic Classes. This could involve the proper mapping of AQUILA TCLs into MPLS "E-LSP" or "L-LSP". Another interesting point that could be investigated is the interaction between the AQUILA provisioning phase and the set-up of MPLS LSP.

As for the VPN service, in a more advanced scenario it could be provided with some form of QoS. The interaction with AQUILA mechanisms is of course possible, but it has to be carefully analysed and specified and this is out of the scope of the implementation work of the

AQUILA project. Two possible interaction scenario are listed hereafter as examples and hints for further analysis:

- A simple scenario is that a VPN is established with a given bandwidth and a specific AQUILA network service (traffic class). It could be seen just as a longer (semipermanent) request to the AQUILA admission control framework.
- Another more advanced scenario foresees that a Best Effort VPN is established, than a specific flow inside the VPN can ask for a QoS reservation to the AQUILA framework.

The theoretical analysis should clarify for the above scenarios the needed admission control mechanisms (how to mix reservation requests coming from VPN with other AQUILA reservations) and traffic control mechanism (how to handle VPN packets and other QoS packets in router queues).

4.2 Content delivery networks

Content Delivery Networks or Content Distribution Networks (CDN) are the answer of network and content providers to satisfy customer demands for high speed and high quality content delivery. A new party, the CDN service provider, seams to be the challenge to make business of the new potential field.

4.2.1 Current Status of CDN

A Content Delivery Network

- is an overlay network to the Internet, covering the higher layers 5 to 7
- is built specifically for high performance of rich multi-media content,
- describes an architecture of web-based network elements.

The mains focus is the support of applications and their delivery to a wide range of customers following their quality requirements. The application focus covers different QoS features. From CDN side the following application groups are identified. In brackets the focus is characterised):

- bandwidth-intensive, rich multi-media content (high resource demands)
- high volume e-commerce transactions (loss sensitive)
- special event, news and entertainment services (large number of customers, multicast)
- interactive video-conferencing sessions (real time aspects, like delay and jitter)

As main components to support the requirements the following three Services are identified:

1. **Distribution Service:** distributes caching, a single hop away from the customer by means of cache servers (surrogates), which are high performance mirrors of the original content.

2. **Redirection Service:** web request redirection on DNS level and load-balancing (web-switching)

3. **Accounting and Billing Service:** distributes collection of usage metering to create added-value services for cash back flows.

There are two industrial groups, both formed in August 2000 to support CDN:

- **Content Alliance:** consortium of 62 partners, enforced by CISCO, aimed on definition, standardisation and implementation of CDN technologies.

- **Content Bridge Alliance:** consortium of 10 partners, founded by Akamai and Inktomi Corp., aimed on prototypical business cases.

4.2.2 AQUILA's CDN Support

As already mentioned above, CDN is an “overlay network” and uses services on layer 7 protocols such as HTTP or RTSP for transport. To deliver content in an optimal way and to give end-users a pleasurable, valuable and consistent experience, many works should be done first including end-user's position, their exploring habits and characteristic of traffic.

Furthermore, if there is too much traffic riding on their network, no matter how huge their backbone segments are, there will be network congestion, latency and packet loss.

In general, the AQUILA project aims to develop a flexible, extendable and scalable Quality of Service architecture for the existing Internet. That means, to support such different traffic characteristics, which arise also in CDNs. As AQUILA is situated in layer 3, naturally it supports upper layered services like CDNs.

The different focus of quality requirements is supported by the AQUILA's Network Service (NS) concept. Offering four premium classes the variety of application's demand is bundled and on router level by means of the Traffic Classes (TCL) optimised. The Network Services are optimised to support typical applications:

- Premium Constant Bit Rate (PCBR) – dialog oriented A/V applications with high jitter sensitivity, like IP telephony

- Premium Variable Bit Rate (PVBR) – dialog oriented A/V applications, like videoconferencing

- Premium Mission Critical (PMC) – online games; e-business transactions

- Premium Multi-Media (PMM) – video on demand

As special applications have concrete requirements, which could be result in a mixture of different optimisation criteria, Application Profiles (AP) support a customising of QoS support using the End-user Application Toolkit.

4.2.3 Conclusions on AQUILA and CDN

As AQUILA covers the lower OSI layers, namely the network layer (L3) and CDNs are looking for support of higher layers L4 (L5) to L7 there is no strong overlapping of both topics. No architectural conclusions can be derived right now. Layer 3 connectivity issues are shifted to intelligent higher layer infrastructures.

The value-added aspects will be covered within the business models of workpackage 3.3 and included in D3301.

4.3 IP multicast

IP multicast is a technique used to distribute IP packets to a number of hosts, called a multicast group, which have registered to receive information sent to this group. There may be more than one sender, which send information to the group.

A multicast routing protocol has the task to distribute information, so that the multicast routers can send a packet from any sender to all receivers. If the senders are not known beforehand, information about the members of the multicast group has to be distributed to all multicast routers in the whole Internet. Multicast routing protocols performing like this are called “dense mode” protocols. The large amount of information however, which has to be exchanged, is one of the major drawbacks of such protocols, especially, when the number of multicast routers grows. For that reason, “sparse mode” protocols are used today. They require however the registration of senders at a so-called “rendezvous point”. Also receivers are required to register themselves using explicit “join” messages. These registrations are used to set up the multicast tree. The simplest case is a sparse mode protocol, when only one sender is involved.

The following discusses the QoS for multicast issue and its relation to the AQUILA approach. A general description is presented with the focus on the intra-domain network view, which is then enhanced to the inter-domain network scenario.

4.3.1 Overview

With the growth of available bandwidth on the last mile up to the end-customers the commercial deployment of broadband services like video/audio conferencing, video/audio streaming or push applications come more and more into the focus of the carriers or ISPs. And in conjunction with the growing bandwidth demand in the network, an IP multicast architecture will get more commercial drive, in order to distribute the large data streams in an efficient way over the IP networks and to save bandwidth. Based on these considerations IP multicast has to be also a topic for the AQUILA project.

The following discussion shows how the AQUILA architecture approach can provide QoS for IP multicast sessions. First of all it must be mentioned that it is not the aim of AQUILA to modify or enhance any multicast mechanisms. The open IP multicast service model is used with the kwon advantages and disadvantages, but as discussed later the multicast protocols have to be enhanced regarding the authentication issue. AQUILA sets up a layer above the multicast distribution architecture for providing QoS for the underlying IP based DiffServ networks. QoS support is only related to the data transmission level.

IP multicast networks offer the service for efficient point-to-multipoint and multipoint-to-multipoint best effort delivery of datagrams. The multicast routing algorithms maintain the connections between the multicast capable routers that form the multicast spanning tree. These trees show the data-paths of the streaming traffic through the network.

Regarding the point-to-multipoint delivery scenario the exemplary graph in Figure 4-5 shows the spreading of the streaming traffic. Each router in the graph shall be multicast capable. One multicast source sends only one data stream into the domain-D at one ingress point. This source is referred to as no-member sender because no data is received. A multicast group exists of different hosts distributed on the network accesses that receive streaming data. A tree with one ingress data stream and many egress data streams describes the data flow.

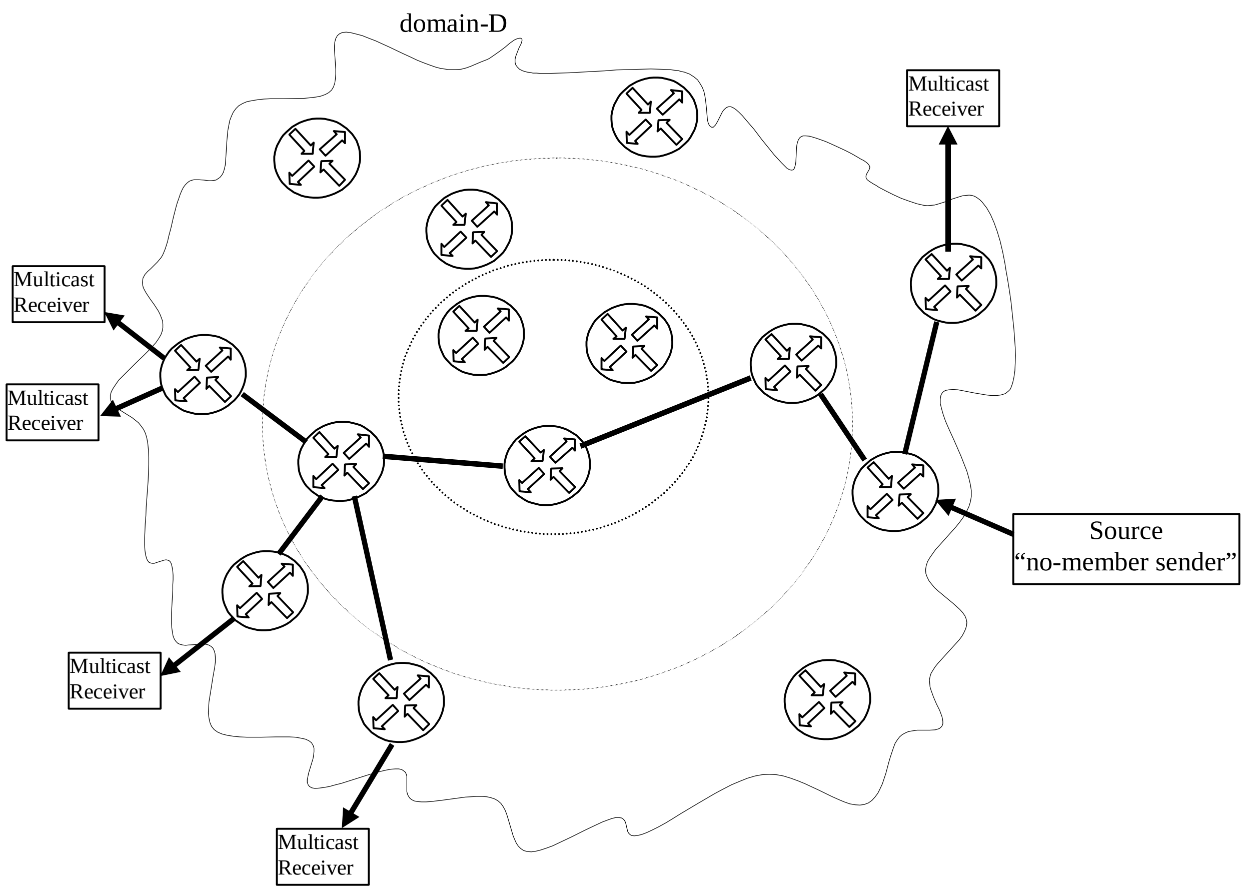

Figure 4-5: Streaming tree of the point-to-multipoint scenario

The point-to-multipoint scenario can mutate to the multipoint-to-multipoint constellation (Figure 4-6). This takes place if more sources send data into the multicast group. E.g. if multicast is used for video/audio conferencing, each multicast receiver also sends data into the multicast group. In this scenario a multicast sender and receiver regarding to one multicast group are hosted on the same entity. The current multicast architecture allows each host to send data into the group, no explicit log on/off mechanism are necessary.

The multicast routing is also an IP addressed based routing like the "usual" routing with the exception that the destination IP address, unlike a single host's IP address, designates a group of receivers. Each multicast capable router decides with an entry in the IP address routing table, on which link the multicast data stream has to be sent to the destination multicast group.

The user's interface to a multicast session is implemented by the "Internet Group Management Protocol" (IGMP) that allows the host to participate at an installed multicast session. Within a multicast domain the PIM-SM, PIM-DM, MOSPF or DVMPR routing protocols organise the intra-domain multicast routing. The inter-domain routing between the multicast domains is task of the MSDP or BGMP protocols. [DeployMCast]

Two different kind of multicast initiations exist:

- The source broadcasts each packet on the ingress network point and the attached multicast-able edge router receives the data and sends it on all interfaces into the network. Each network router will perform the Reverse Path Forwarding (RPF) check and accepts the packets only if a neighboured router doesn't reject the multicast stream. Otherwise the router signals the rejection backward with the PRUNE message. In this way reverse shortest path trees are created with so called **Broadcast-and-Prune** protocols also known as **dense mode** protocols. The disadvantage is that the state information must be kept for each source at every router in the network regardless if group member exist.

- The other kind of multicast initiation is based on the **sparse mode**. The protocols designed in this mode don't broadcast the traffic into the network and don't send prune messages. The sources are expected to send the traffic to a meeting node in the core a so-called rendezvous point (RP). The receivers send explicit joint messages to this RP.

In order to participate in a multicast group two main steps are necessary from the host's point of view:

1. When a host wants to join a new multicast group, it sends an IGMP message to the group's multicast address in order to signal the membership interest. The local multicast router receives the IGMP message and establishes the necessary routing by propagating this information to other neighboured multicast routers with the above described sparse or dense mode protocols. The IGMP protocol only handles the host router interface.

2. After this initialisation step the local multicast router queries the current multicast state of the hosts, and if no hosts of this routing have stored a membership in this multicast group, this router removes the stored routing information and stops the advertising group membership to other multicast routers.

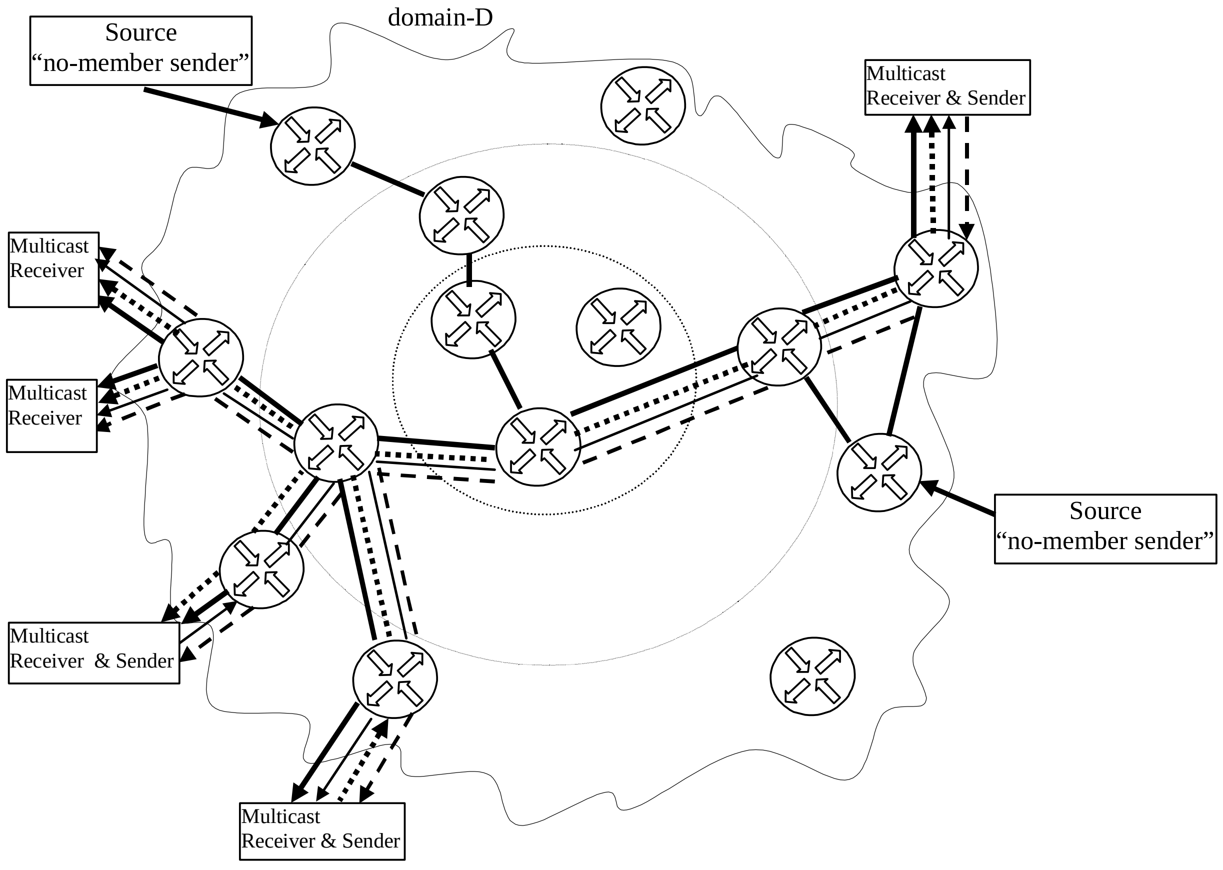

Figure 4-6: Streaming tree of the multipoint-to-multipoint scenario

4.3.2 QoS support for IP multicast in AQUILA domains

In the following only the topic “QoS support for IP multicast within a domain” is discussed and therefore assumed that one autonomous system is the border for the multicast traffic. Inter-domain issues are covered in the chapter 4.3.3.

The AQUILA architecture is based on the prerequisites:

- At each ingress point in the network an edge device is placed for policing, marking and scheduling the incoming data streams.
- A reservation request has to be done for each QoS traffic flow that will be sent into the network, otherwise the prioritisation of the traffic stream is remarked to best effort.
- The network itself is based on a DiffServ architecture with different traffic classes, and only policed data streams share the bandwidth allocated for these traffic classes.

4.3.2.1 Scenario point-to-multipoint

4.3.2.1.1 Requirements

In the point-to-multipoint scenario (Figure 4-5**)**, only one content provider originates a multicast session, and streams his e.g. video/audio content into the multicast aware network. Each user has the possibility to join this multicast session.

- For this type of multicast session only **one** source is allowed to send into the network.
- The network egress points of the multicast traffic have to be recognised in order to be able to set up resource reservations at this egress point.

Therefore the AQUILA RCL needs the information which host enters the QoS multicast group at which edge device. This is needed in order to evaluate if the network is able to transfer the QoS traffic stream with the requested network service up to the egress edge device.

Therefore the multicast protocols should support a mechanism to manage group membership and to propagate this information to an upper layer like the AQUILA RCL. The multicast research enters this path but in the moment nothing like this is available in commercial router equipment.

4.3.2.1.2 AQUILA oriented proposal

The following discussion shows a solution how the AQUIAL RCL can handle QoS for **closed** or **open** multicast groups under the mentioned conditions. A QoS reservation for a multicast session can be split into two steps.

- a source based reservation request at the network ingress point
- a receiver based participation request at the network egress point

<u>First Step: Reservation Request at the ingress point of the multicast stream</u>

If the source will offer a QoS based multicast session a reservation request to the EAT (End-user Application Toolkit) has to be done for this multicast UDP traffic stream. The multicast destination IP address is not useful to determine an egress point of the network. And in the current state of the multicast architecture the source has no information about the multicast members at this session.

However, the multicast traffic stream that enters the network is distributed over the network (dense mode) or forwarded to a rendezvous point (sparse mode). The router at the RP will discard the received data, because in the initial phase there are no registered receivers.

Therefore the reservation request at the egress point of the network has to be handled in a second step independent from the source based reservation request.

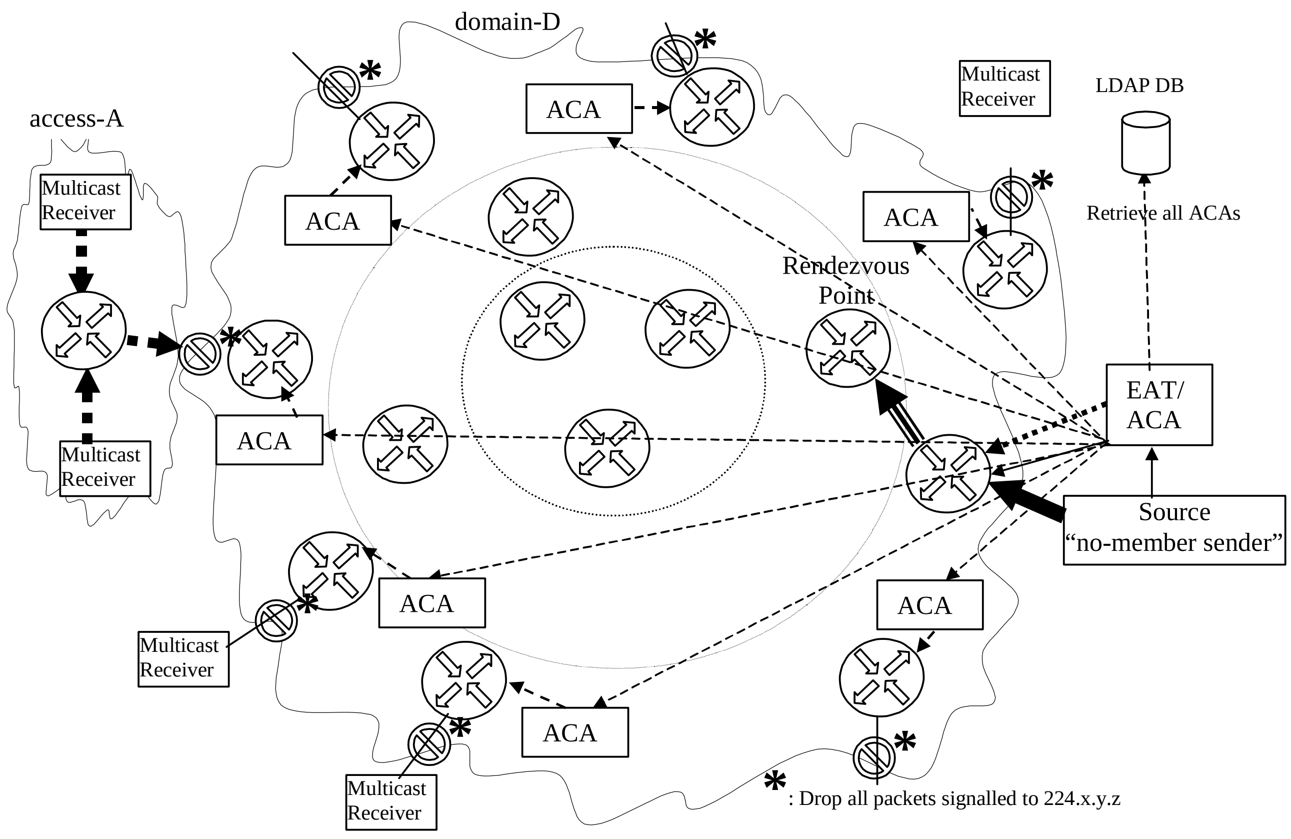

Figure 4-7: Reservation Requests for the multicast session

Reservation procedure to set up a closed / open QoS multicast group

- Therefore the reservation request form the source to the EAT has to include the information that this QoS multicast session is initiated for a closed group with the IP multicast address 224.y.x.z.

- The EAT transfers this information to the ACA manager that is responsible for this ingress edge device (ED) and that has to set up a closed QoS multicast group.

- The responsible ACA retrieves a list from the database with all ACAs that are installed in the domain-D.

- In the next sub-step the ACA manager initiates at each ACA the installation of a policer for this QoS multicast IP address 224.x.y.z with the condition that no package signalled to this group has the permission to enter the network.

  The ACA manager has to transfer the traffic specification to all ACAs in order to provide the egress side with the needed information if a receiver host signals its participation request independent from the source reservation.

- Finally the ACAs at the egress side have to inform the corresponding EATs at the egress side that this new multicast group is available inside the domain-D.

Second Step: Participation Request at the egress point of the multicast stream

Closed QoS multicast group

Each user has to be registered otherwise participation at this multicast group isn't possible.

Open QoS multicast group

In this scenario the main prerequisite is that all user can participate without an explicit authentication procedure. The content provider offers a free multicast stream with a good QoS e.g. an advertisement video and pays himself for the costs. Therefore a default rejection of all signalling traffic or data traffic to this multicast group isn't required in order to authenticate the user.

But the AQUILA resource control layer has to check whether an additional QoS data stream can be carried through the network to this especial edge device. However, it is a postulation of the AQUILA RCL, that in any case the host's participation at an open QoS multicast session has to be signalled to the EAT responsible for this egress point.

If the QoS request is rejected it is not possible for the AQUILA RCL to set up a best effort multicast stream to this egress point as alternative. Because a remarking in the core would be necessary and the multicast router, that has to perform this, would change dependent on which access area is connected in which constellation to this QoS multicast group. So if the participation request to a QoS multicast group is rejected the host don't receive any data traffic of this session QoS session.

Therefore the open group scenario is identical to the closed group scenario with the exception that the accounting information related to the QoS multicast participation request isn't bound to an authenticated user.

Procedure:

- In the second step the user is forced to order at the EAT the participation for this QoS multicast session. No reservation style is necessary because the receiver has no information which traffic characteristic information is needed. This information is already stored at the ACAs (see Step 1).

  Now the authentication control part of the AQUILA RCL performs the authentication mechanism for the multicast "layer". And if the user is allowed to participate at this multicast group, the new receiver becomes in the next step the possibility to send multicast signalling data into this group.

  Nevertheless the AQUILA RCL needs the information in any case who is a member of the receiver-group.

- Initiated by the users reservation request, the EAT signals the ACA manager, responsible for this egress edge device, that the installed policer for this QoS multicast group (address 224.x.y.z) can be modified. Now only signalling information is allowed to forward into the multicast group. But a data traffic stream into the multicast group has to be discarded further on. The accounting information has to be coupled to the authenticated user.

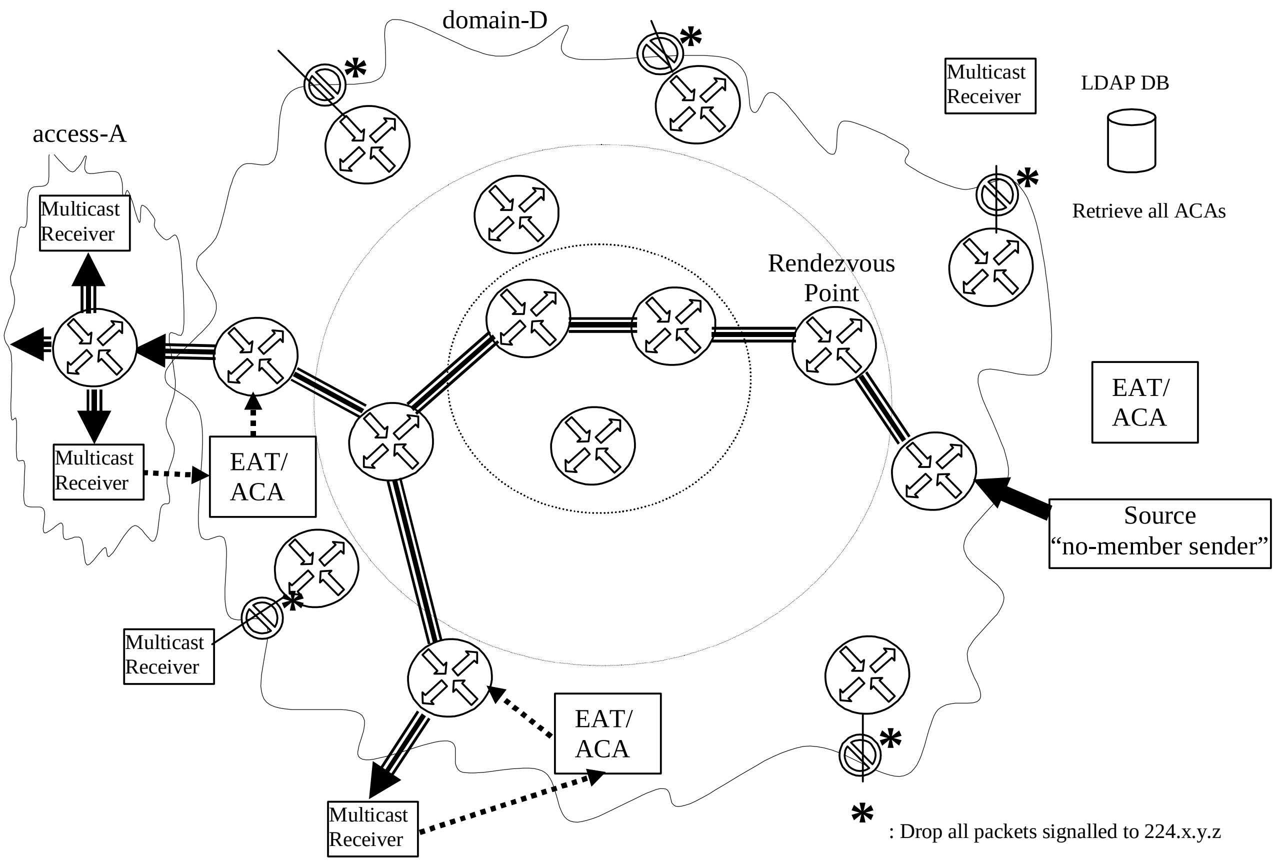

Figure 4-8: Participation Requests to an established multicast session

The disadvantage of these scenarios is that now the whole access area-A can receive the QoS multicast stream if only one hosts of this area sets up a reservation request to a closed multi-cast group. But this is a general problem of the multicast architecture and not resolvable from the AQUILA RCL.

4.3.2.2 Scenario multipoint-to-multipoint

In the multipoint-to-multipoint scenario more sources can send into the network, independent if the sender is also a receiver or only a "no-member sender" (see Figure 4-6).

This complex multicast scenario can be split into several point-to-multipoint sessions Figure 4-9 as discussed in chapter 4.3.2.1.

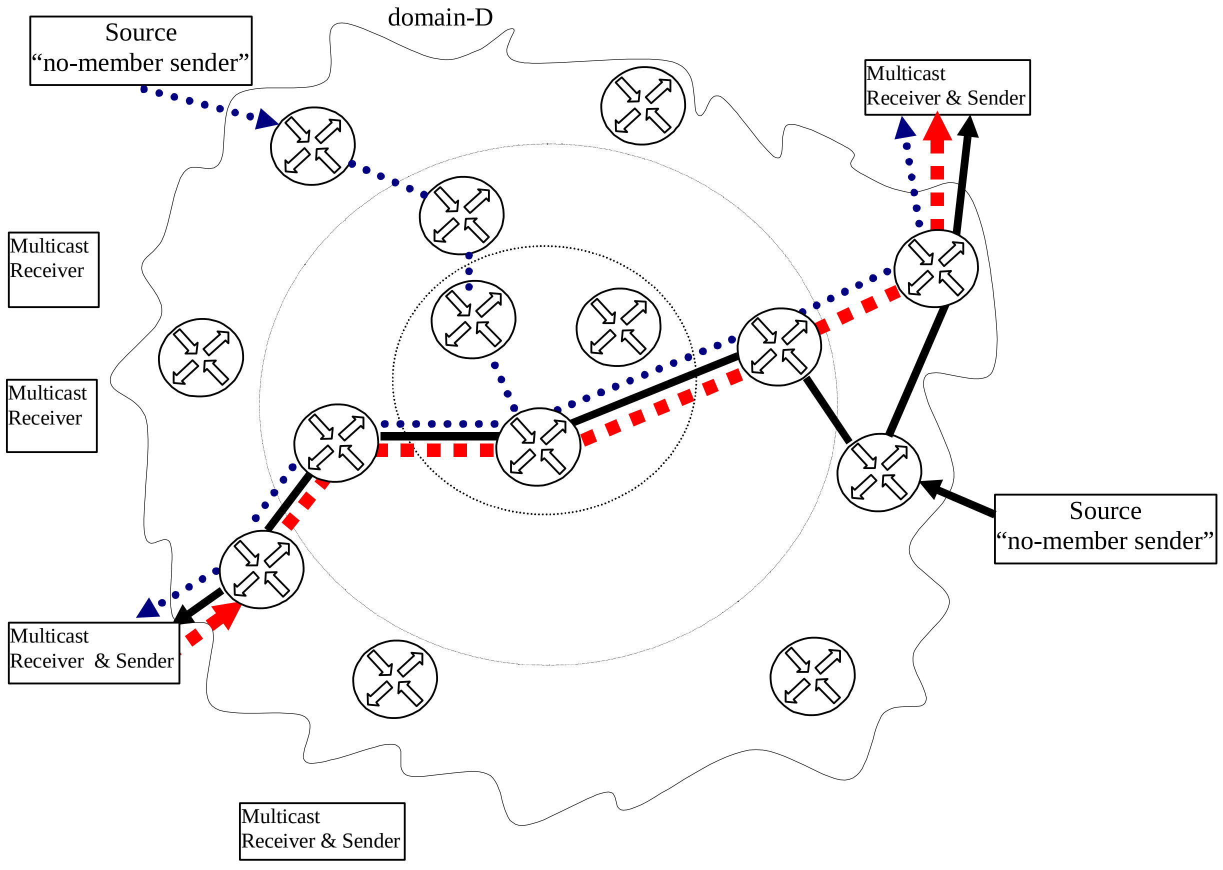

Figure 4-9: Multipoint-to-multipoint session split into several point-to-multipoint sessions

But in general the multipoint-to-multipoint includes more complexity:

First of all, the scenario illustrated in Figure 4-5 can be split into three point-to-multipoint scenarios, because three sources exists and each source has to set up a QoS multicast reservation request independent if the source is also a receiver.

Nevertheless when the first “no-member” sender starts the reservation request then the principal procedure handling is identical as described in chapter 4.3.2.2. But some small extensions are important.

1) If more than one source should be allowed to send into the same multicast group each sender has to be authenticated as a member of the sender-group.

   The authentication mechanism is important for an undisturbed functionality of this multipoint-to-multipoint multicast session but without any QoS specific background and therefore not so important for the AQUILA itself.

2) A major topic for the AQUILA RCL is the situation if all sources and receivers have successfully requested resources and then one source logs out or is unavailable. The question

raises up how the other multicast group members are effected. Should only the reservations for this source at the ingress and the bundled reservations at the egress side be cancelled or all other reservation parts of multicast session?

Only an initiator or administrator of this session or the sources itself can decide this question. Therefore it is important for the resource handling in the AQUILA domain that at least the source reservation request at the EAT includes the information if this source is essential for the whole multicast session or not.

Then the AQUILA RCL can decide which resources have to be released and how the group members can be informed.

3) If a receiver starts its participation request at a multicast session with several active sources, it has to be in mind that now at this egress edge also several resource reservations have to be done. In this constellation one participation request at the receiver side initiates several reservation at this edge device.

4.3.3 Inter-domain multicast reservations

When considering multicast inter-domain resource control, primarily the same arguments apply, which are already presented in chapter 3.2 of this document. While resource control for a single domain can be applied by just controlling the ingress and egress, the inter-domain scenario requires considering each domain and the links between them.

The following sub-chapters roughly describe the state of the art for inter-domain IP multicast routing and propose a coarse framework for a possible solution.

4.3.3.1 Inter-domain IP Multicast

As in the unicast case, inter-domain multicast reservations always have to follow the routing paths set up by the routing protocol. For unicast, BGP is the routing protocol used today in the Internet. For multicast however, routing protocols are still evolving.

While the first multicast network, the so-called MBone, assumed a flat topology formed by tunnels connecting multicast routers, this flat topology becomes more and more unmanageable as the network grows. So the need for hierarchically inter-domain routing protocols evolved.

Today, a solution combining several protocols is used, called MBGP/PIM-SM/MSDP.

- MBGP [RFC2283] is an extension to BGP, which is able to carry multicast routes. This information may be used to construct a multicast tree using a reverse path back to the source. However, MBGP does not construct such trees itself.

- PIM-SM [RFC2362] (protocol-independent multicast – sparse mode) is a protocol providing multicast tree construction functions.

- MSDP [MSDP] is the multicast source discovery protocol used to announce multicast sources to neighboured domains.

4.3.3.2 Towards inter-domain multicast reservations

An inter-domain resource reservation protocol has to follow the paths constructed by whatever routing protocol is used in order to reserve resources on those links, which are also used by the actual data. This is one of the principles of the BGRP unicast resource reservation protocol, but also holds for multicast resource reservations. However, multicast makes things much more complex, because now any number of senders and any number of receivers have to be considered, and traffic is flowing from all the senders to all receivers of a multicast group.

Note that it is difficult to propose a multicast reservation architecture, when the properties of the multicast routing protocol are still unknown. Nevertheless in the following a coarse idea of a multicast resource reservation scheme is sketched. The basic idea is to break up the multicast reservation into hop-by-hop reservations along the multicast tree and use an existing unicast inter-domain reservation protocol for each hop. Some basic ideas of BGRP are re-used and adapted to the multicast scenario:

Each sender of a multicast group initiates a resource request for the traffic, that he will send to the group. The request will use the sender's address as source address and the multicast address as the destination address. This message corresponds to the PROBE messages of BGRP.

Each hop checks, whether it knows any downstream multicast group members. If no, there is no need for this reservation and the request is rejected. However, if there are any downstream multicast group members, each hop immediately reserves the resources answers the request, acting, as if it would be the sink of the data. It then initiates requests to all next hops for that multicast group.

As the requests are forwarded through the network in this way, they are distributed to all receivers of the multicast group. This implicates that all receivers in a multicast group will share the same reservation, and that it is not possible to have QoS receivers and best effort receivers at the same time in the same multicast group.

During the lifetime of the multicast reservations, refresh messages are sent periodically. These messages will discover the addition or removal of group members and adjust the reservations accordingly.

Note however, that because the sender (and initiator of the reservation) does not know the receivers, he can also not be informed about any failure in the reservation. Instead, the affected receivers may be notified, if a reservation fails on some part of the multicast tree.

The proposed mechanism actually splits up a multicast reservation into a number of hop-by-hop reservations, which are arranged in a way, so that they resemble the multicast tree. The mechanism does not require any change in the multicast routing protocol used. However it requires, that it can get the list of next hops for a multicast address from the routing protocol, just as BGRP requires to get the NEXT_HOP attribute from BGP.

4.4 Security

It is not a main topic in AQUILA project to investigate security concepts for the proposed structure. This Chapter is just a study, and its aim is to demonstrate that the proposed architecture could be a secure network, and to show a way it could be achieved.

The AQUILA architecture is constructed as an overlay to a DiffServ aware network. The aim of all attacks to QoS provisioning of DiffServ networks is stealing not provisioned resources or damaging the QoS (denial of QoS) of other traffic [DSSEC]. The higher price associated with the enhanced services creates incentives to steal.

4.4.1 Secure Signalling

The signalling traffic is a vulnerable part of the architecture because of its functionality as a QoS administrator. Due to the AQUILA overlay network, signalling traffic occurs within the RCL, between the user and the RCL, between the RCL and the DiffServ network and between different domains.

4.4.1.1 Signalling within the RCL (intra-domain)

The Resource Control Layer is a logical layer composed by software components that share the physical network. It does not exist a protected network for the signalling traffic.

Each RCL component owns a CORBA interface, which enables the communication between all the components. The rather general structure of the CORBA security service is flexible enough to support a wide variety of security policies, but results in a rather complex architecture. In order to size up the grade of security needed by the architecture is necessary to analyse how this signalling is working. The intra-domain signalling interchanges messages between the RCL components from the same domain, this enables an uncomplicated security policy, mainly based on authentication, to cover the general security requirements.

To be conformant to CORBA Security Service (CORBASEC) Specifications [CORBASEC], the security policy for intra-domain signalling could use the CORBASEC Security Level 1:

CORBASEC Security Level 1

- provides a first level of security for applications which are unaware of security and for those having limited requirements to enforce their own security in terms of access controls and auditing.

CORBASEC Security Level 2

- provides more security facilities and allows applications to control the security provided at object invocation. It also includes administration of security policy, allowing applications administering policy to be portable.

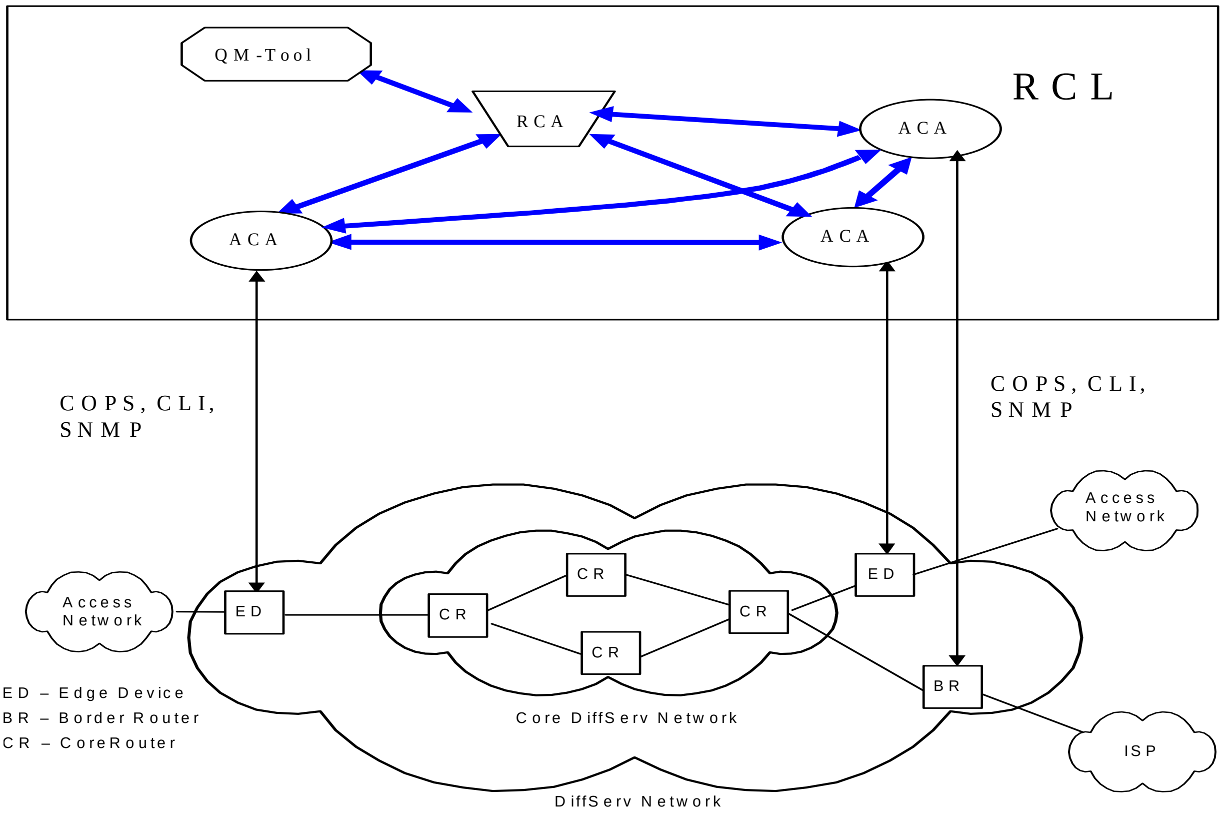

Figure 4-10: Signalling traffic within the RCL

More in detail the Level 1 covers authentication of principals (provided by SESAME authentication protocol), secure invocation between client and server involving trust, encryption and ACLs (Access Control Lists), Delegate incoming credentials and Auditing.

In this first approximation to the security in the AQUILA architecture a combination of ACLs (Access Control Lists) and authentication of principals is proposed for the intra-domain signalling.

4.4.1.2 Signalling traffic into and out of the AQUILA domain

4.4.1.2.1 Signalling between different domains (inter-domain)

The inter-domain resource control signalling is performed by an additional layer built on top of the intra-domain resource control and it is independent of the implementation of the intra-domain resource control, so that network operators are free to use any kind of resource control within their domain. In 3.2. the BGRP framework is proposed for AQUILA.

BGRP is an inter-domain reservation protocol. Reservation aggregation is at the domain level and only BGP-speaking routers in the network participate in reservation process. BGRP runs over TCP.

The vulnerability of the information transported over BGRP recommends the use of security protocols such as SSL or TSL.

4.4.1.2.2 Signalling traffic from the user to the RCL

The signalling traffic correspondent to the communication from the user to the EAT (End-user Application Toolkit) needs to be considered separately to the communication within the RCL. In this case, we can not consider any longer the signalling to be in a trusted domain, and security protocols as SSL/TSL need to be considered.

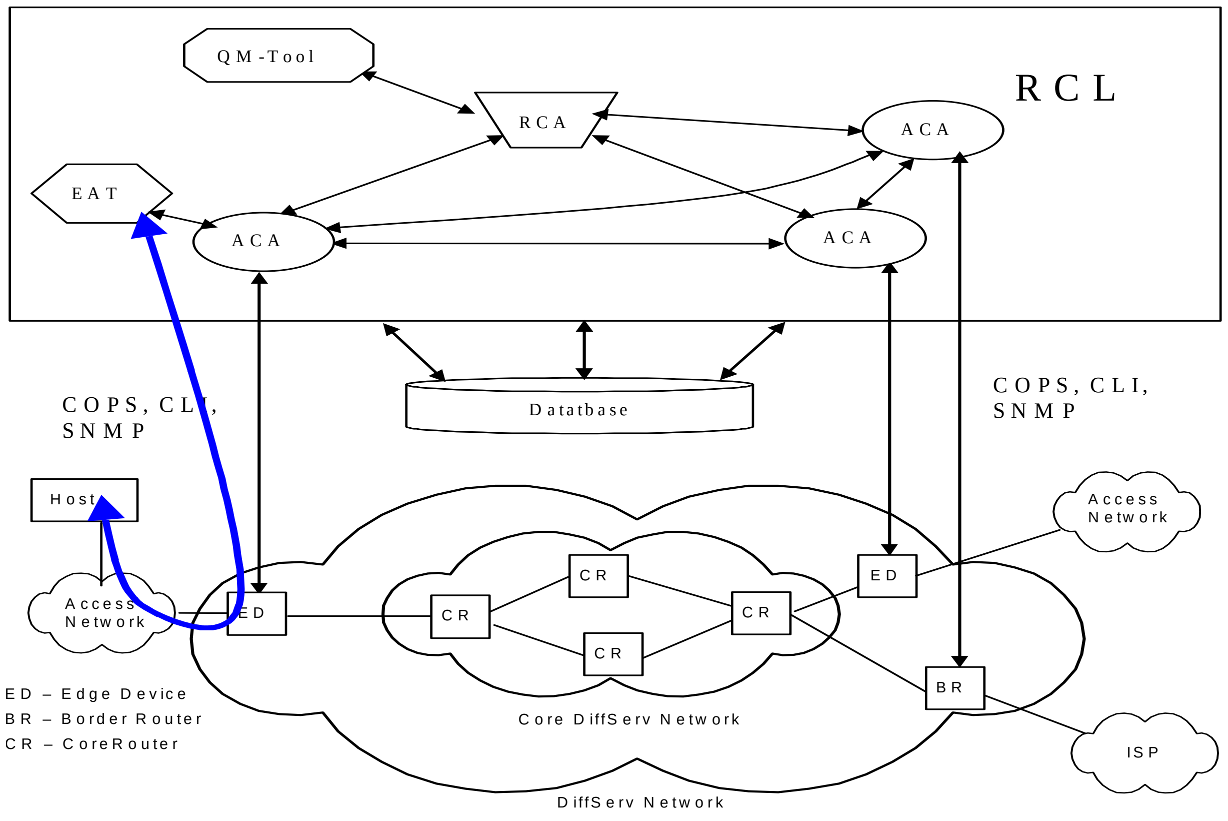

Figure 4-11: Signalling traffic from the user to the RCL

4.4.1.3 Signalling between RCL and DiffServ network

For the communication between edge devices and the core routers and the ACA as part of the RCL no middleware like CORBA is used. The ACA configures the routers via the Command Line Interface (CLI) or with Common Open Policy Service (COPS). The CLI use a TELNET connection to the router.

SNMP is another protocol to control routers. But with SNMP only a limited number of instructions can be exchanged. SNMP is only used for the configuration of routers in conjunction with a configuration tool. But anyhow, TELNET and SNMP are the two application protocols for the configuration of routers.

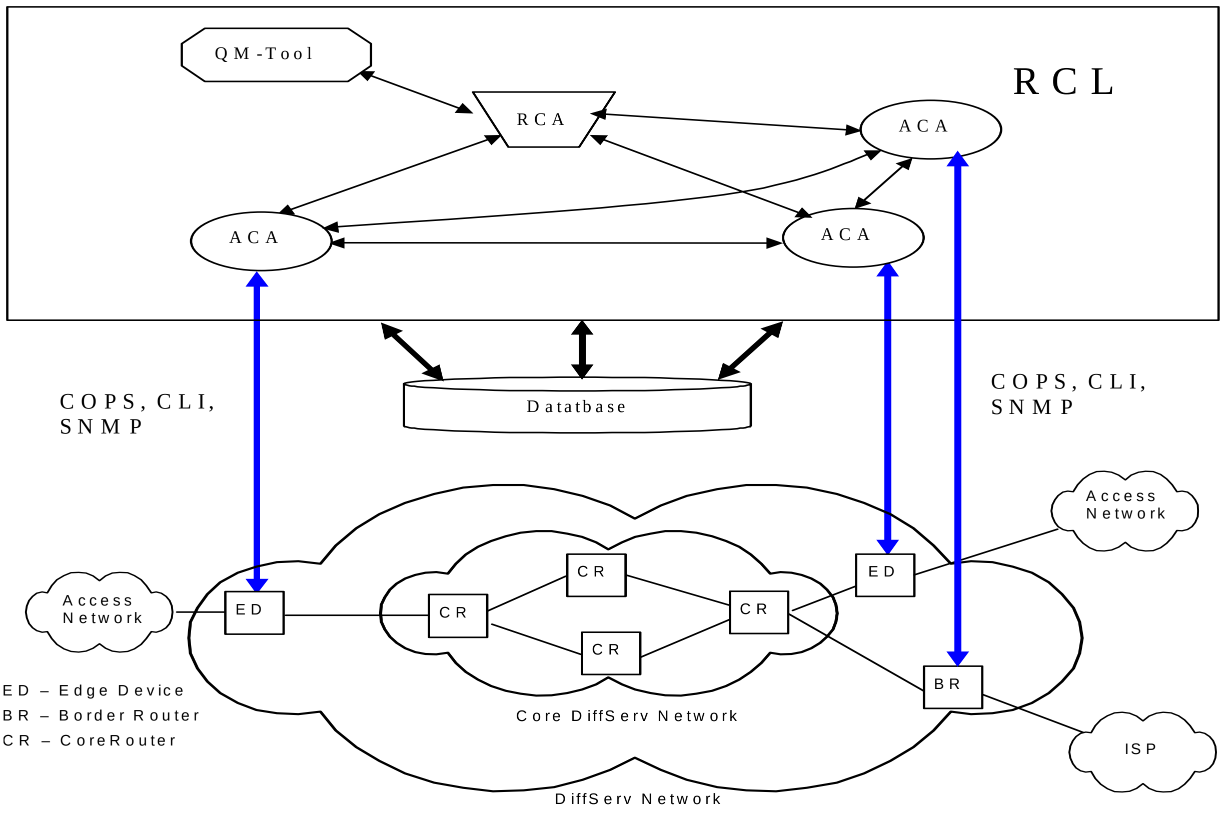

Figure 4-12: Signalling traffic from the RCL into the DiffServ domain

TELNET and SNMP are both insecure. TELNET transfers the User-Mode and Privileged-Mode Passwords in plain text. Secure Shell (SSH) is a protocol that provides a secure, remote connection to a router. There are currently two versions of SSH available, SSH Version 1 and SSH Version 2. Only SSH Version 1 is implemented in Cisco IOS.

The SSH server feature enables a SSH client to make a secure, encrypted connection to a router. This connection provides functionality that is similar to an inbound Telnet connection. The SSH server in Cisco IOS will work with publicly and commercially available SSH clients.

Before SSH, security was limited to Telnet security. SSH allows strong encryption to be used with Cisco IOS authentication.

4.4.1.3.1 Restrictions

SSH imposes the following restrictions:

- RSA authentication available in SSH clients is not supported in the SSH server for Cisco IOS.
- User ID and Password authentication only.
- Supported on DES (56-bit) data encryption and Triple DES (168-bit) data encryption software images only. In the DES (56-bit) software images, DES is the only encryption algorithm available. In the Triple DES software images, both DES and Triple DES encryption are available.
- Execution shell is the only application supported.

Supported Cisco Platforms:

- Cisco 7200 series routers
- Cisco 7500 series routers
- Cisco 12000 series Gigabit Switch Routers (GSR)

4.4.1.3.2 Common Open Policy Service

The Common Open Policy Service is a simple client/server model for supporting policy control over QoS signalling protocols. The COPS protocol runs over TCP. It is a query and response protocol that can be used to exchange policy information between a policy server (Policy Decision Point or PDP, in AQUILA the ACA) and its clients (Policy Enforcement Points or PEPs, in AQUILA the ED).

COPS is responsible for the authorisation of access to network resources and is responsible for the provisioning of network resources. The ED must trust it is communicating with the correct "master". COPS runs over TCP allowing a number of connection security choices:

- IPSec requirement (see 4.4.3.2)
- COPS Integrity Object requirement
- TLS socket level security requirement

COPS Integrity is the original security solution for COPS in [RFC2748]. There should be additional COPS Integrity objects that provide:

- Authentication
- Replay prevention

- COPS message integrity validation

The COPS Integrity solution will be part of COPS and therefore provided by all COPS implementations, regardless of underlying TCP/IP stack. It allows COPS to actually be aware and in control of the security of its communication. The Integrity is a firewall and NAT friendly mechanism and IPSec or Secure Sockets mechanisms can still be used as well (as an option)

4.4.1.3.3 Access to the router

Anyone who can log into a router can display information, which AQUILA, probably, does not want to make available to the general public. A user who can log in to the router may be able to use it as a relay for further network attacks. Anyone who can get privileged access to the router can reconfigure it. To prevent inappropriate access, AQUILA needs to control interactive logins to the router.

There are more ways of getting interactive connections to routers than users may realise. Cisco IOS software, depending on the configuration and software version, may support connections via Telnet; rlogin; SSH. All interactive access mechanisms use the IOS TTY abstraction. There are not only physical ports or interfaces to connect for the configuration of the router. There are also virtual ports, which are called vty (by default five such virtual terminal lines). Standard and extended access lists will block packets from going through the router. They are not designed to block packets that originate from the router. An outbound TELNET extended access list does not prevent router-initiated TELNET sessions.

For security purposes, users can be denied vty access to the router, or users can be permitted vty access to the router but denied access to destinations from that router. Restricting vty access is less a traffic control mechanism than a technique for increasing network security.

Disabling all non-IP-based remote access protocols, and using IPSec encryption for all remote interactive connections to the router can provide complete VTY protection.

Denial-of-service attacks are relatively common on the Internet. If the network is being subjected to a denial of service attack, the administrator may not be able to reach the router to collect information or take defensive action. Even an attack on someone else's network may impair the management access to the own network. AQUILA can take steps to make the network more resistant to denial of service attacks, the only real defence against this risk is to have a separate, out-of-band management channel, such as a dialup modem, for use in emergencies.

4.4.2 LDAP Database

The communication between the RCL and the network includes the communication with management components of the DiffServ network like databases, accounting post-processing entities, policy server etc. The configuration properties of the AQUILA RCL are stored on LDAP database and dynamically used from all the RCL components. Therefore, a security mechanism is needed for this communication.

The LDAP Database needs to be accessible only by the RCL components. The modifications by anonymous users must be prevented through authentication. The server may also minimise denial of service attacks by timing out idle connections. The LDAP version 3 offers ways to access a rich set of security functions [RFC 2829].

According to the specifications, LDAP protocol suite can be protected with the following security mechanisms:

(1) Client authentication by means of the SASL mechanism set, possibly backed by the TLS credentials exchange mechanism,

(2) Client authorisation by means of access control based on the requestor's authenticated identity,

(3) Data integrity protection by means of the TLS protocol or data-integrity SASL mechanisms,

(4) Protection against snooping by means of the TLS protocol or data-encrypting SASL mechanisms,

(5) Resource limitation by means of administrative limits on service controls, and

(6) Server authentication by means of the TLS protocol or SASL mechanism.

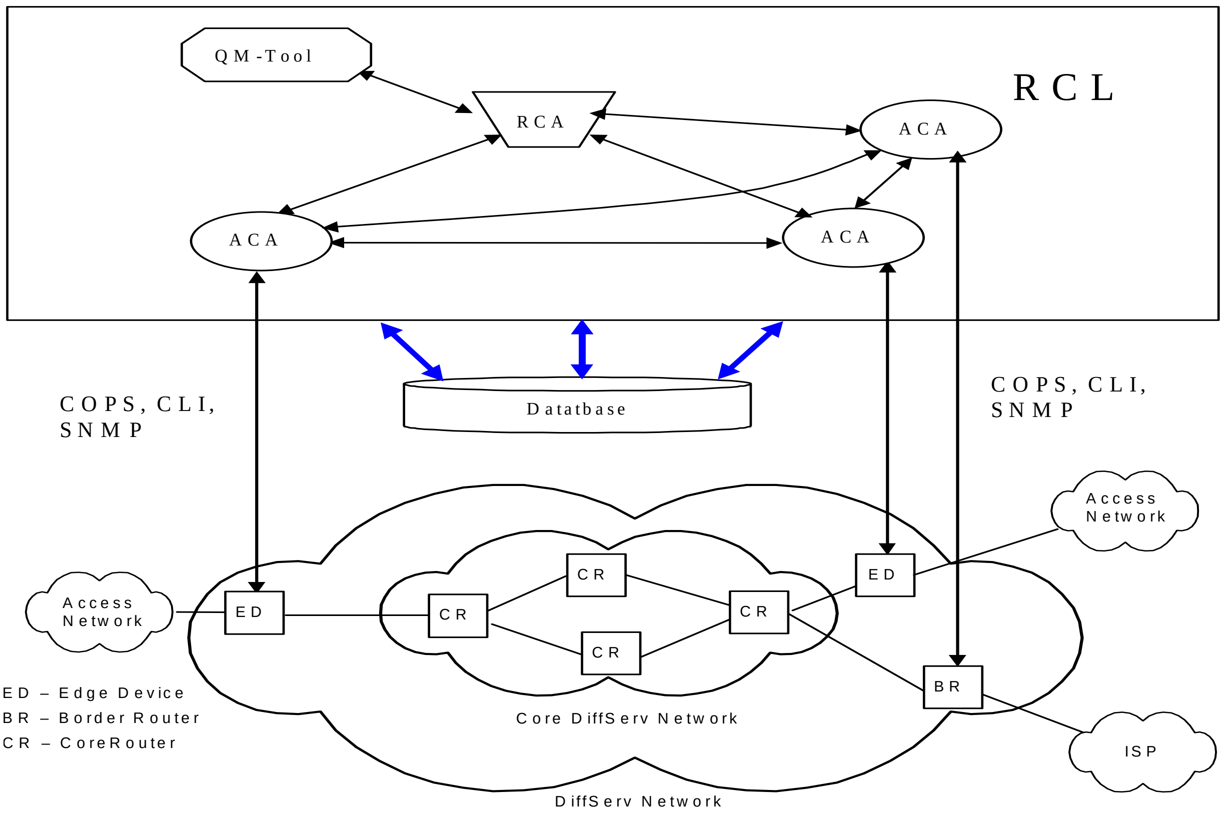

Figure 4-13: Communication with the LDAP Database

Active intermediary attacks are the most difficult for an attacker to perform, and for an implementation to protect against. In the AQUILA architecture the risk of active intermediary attacks could be considered as a low risk and does not justify the cost of protection against active intermediary attacks. Methods that protect only against hostile client and passive eavesdropping attacks are recommended.

4.4.3 Security in the DiffServ network

The AQUILA network guaranties different level services to be provided for traffic streams on a common network infrastructure. The mapping of network traffic to the specific behaviours of different services is indicated primary by the DiffServ code point (DSCP).

4.4.3.1 Theft and Denial of Service

An adversary maybe able to obtain better service by modifying the DSCP to values indicating behaviours used for enhanced services. These theft-of-service become a denial of service attacks when the modified or injected traffic depletes the available resources [RFC2474]. The defence against this class of theft- and denial-of-service attacks consists of the combination of traffic conditioning at the network boundaries and integrity of the network infrastructure. Therefore the ED must ensure, that all traffic entering the domain is marked with DSCP.

Any traffic-originating node within a DiffServ domain is the initial boundary for the traffic. The boundary nodes are the primary line of defence against theft- and denial-of-service attacks. In the AQUILA architecture the ED take the task of the boundary node.

The AQUILA network is connected over a Boarder Router (BR) to another network. The BR also has to handle theft- and denial-of-service attacks. Cited from [BGRP]:

> *In the BGRP model, we always assume some level of trust between BGRP routers. The reservation information is delivered domain by domain. Without proper authentication, this will enable denial of service attacks. Integrity information is required for each BGRP message.*

Interior nodes in a DiffServ domain rely on DSCP to associate traffic. Therefore the interior nodes are within a trusted domain. Every traffic-originating node (excluding multicast routers) is controlled, either by the ED or the BR.

4.4.3.2 IPSec and Tunnelling Interactions

IPSec uses two protocols to provide traffic security Authentication Header (AH) and Encapsulating Security Payload (ESP).

The IP Authentication Header (AH) [RFC2402] provides connectionless integrity, data origin authentication, and an optional anti-replay service.

The Encapsulating Security Payload (ESP) protocol [RFC2406] may provide confidentiality (encryption), and limited traffic flow confidentiality. It also may provide connectionless integrity, data origin authentication, and an anti-replay service. (One or the other set of these security services must be applied whenever ESP is invoked.)

Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

To classify flows the Admission Control Agent (ACA) uses port numbers. The ESP protocol encrypts the TCP header. The source and destination port is no longer visible for the ACA. Encrypted data streams have to be pre-marked with the equivalent code point. The ACA can still control the flow and only the classification is different.

4.4.4 User Authentication

In the AQUILA network the user request the QoS for a single flow over the EAT or the WEB server. There are a lot of mechanisms for Authentication and Authorisation. In the IETF exist two protocols in the AAA working group: DIAMETER and RADIUS.

The DIAMETER Strong Security extension currently only supports Cryptographic Message Syntax (CMS) in an asymmetric mode. Kerberos has been proposed as an alternative to establish dynamic security associations (keying material) between DIAMETER peers. The Diameter base protocol leverages either IPSec or TLS for integrity and confidentiality between two

Diameter peers, and allows the peers to communicate through relay and proxy agents [CMS]. The Diameter protocol also allows peers to communicate through relay and proxy agents, and in such environments security information is lost at each agent.

The RADIUS protocol was developed by Livingston Enterprises, Inc., as an access server authentication and accounting protocol. Implemented by several vendors of network access servers, RADIUS has gained support among a wide customer base, including Internet service providers (ISPs). The RADIUS authentication protocol [RFC2058] is documented separately from the accounting protocol [RFC2059], but the two can be used together for a comprehensive solution.

Cisco has developed an own authentication and authorisation protocol. The Cisco TACACS+ protocol uses TCP instead of UDP like RADIUS. RADIUS encrypts only the password in the access-request packet, from the client to the server. The remainder of the packet is in the clear. A third party could capture other information, such as username, authorised services, and accounting.

TACACS+ encrypts the entire body of the packet but leaves a standard TACACS+ header. Within the header is a field that indicates whether the body is encrypted or not. For debugging purposes it is useful to have the body of the packets in the clear. However, normal operation will fully encrypt the body of the packet for more secure communications.

TACACS+ uses the AAA architecture, which separates authentication, authorisation, and accounting. This allows separate authentication solutions that can still use TACACS+ for authorisation and accounting. For example, with TACACS+, it is possible to use Kerberos authentication and TACACS+ authorisation and accounting. After a NAS authenticates on a Kerberos server, it requests authorisation information from a TACACS+ server without having to re-authenticate. The NAS informs the TACACS+ server that it has successfully authenticated on a Kerberos server, and the server then provides authorisation information.

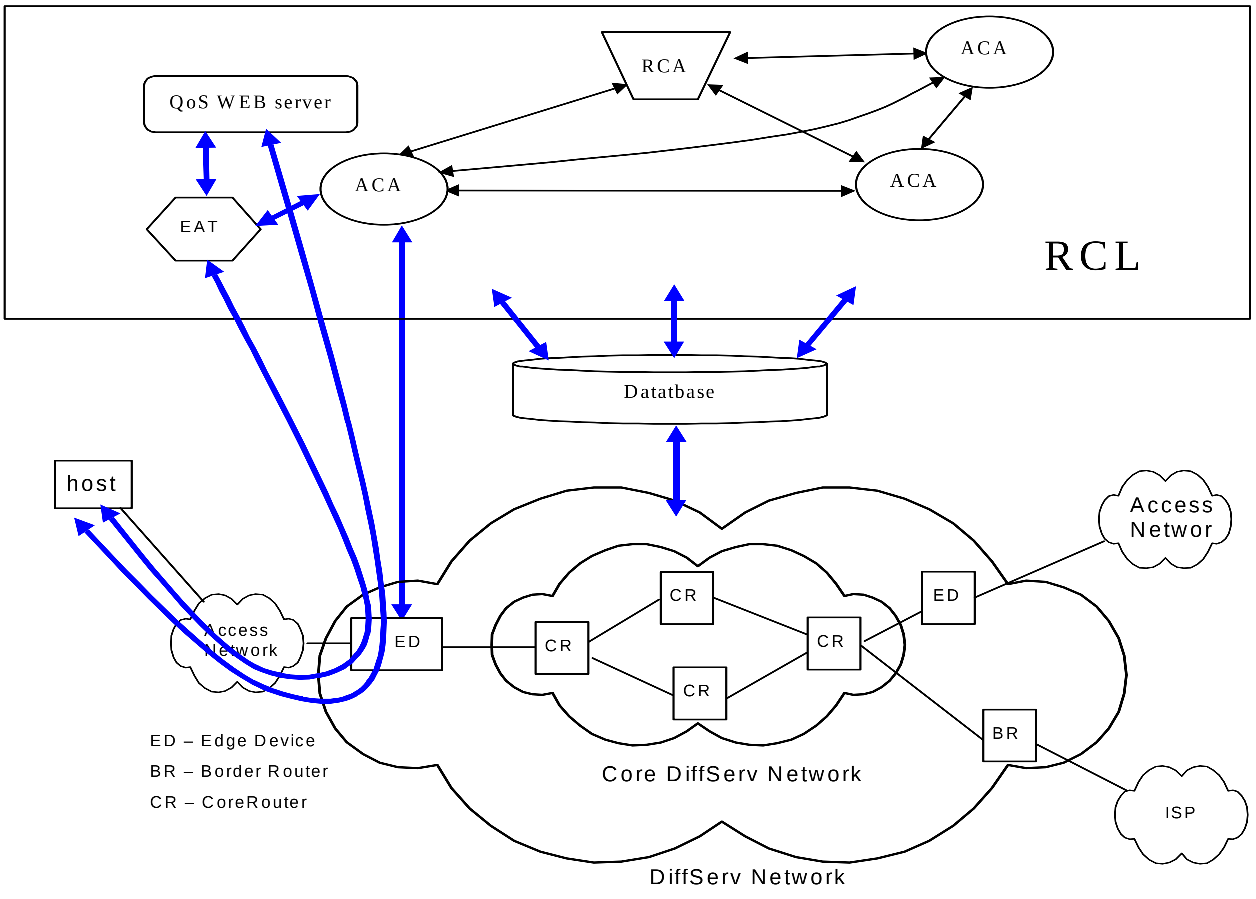

Figure 4-14: User Authentication

End-to-End is the security model that requires that security information be able to traverse, and be validated even when an AAA message is processed by intermediate nodes such as proxies, brokers, etc.

4.4.4.1 Database

The data of each user like the SLA with the provider is stored in the LDAP database. For the security aspects of the LDAP database see 4.4.2.

The database with the user data is not necessarily the same database as in the current architecture. The database of the configuration properties of the RCL and the database with the user data can logically divided in two separate databases. Therefore, a security mechanism is needed for the database with the user data.

The LDAP Database needs to be accessible only by the EAT. Modifications by anonymous users must be prevented and the server may also minimise denial of service attacks by timing out idle connections. Only the ISP has access to the database and the database has to be separated from the network.

The EAT gets the user information from the database and stores the data during a session. The accessibility of the EAT must be prevented. The security demands are the same as for the other RCL components (see 4.4.1.1).

4.4.4.2 Authentication

The EAT provides a Log In/Logs Out mechanism for each end-user. The login is for identification of the end-user and to be able to use the EAT as an personal account needed by every user. In this manner it is possible to store specific end-user data, and to offer a personalised EAT.

The EAT Manager provides a traditional login procedure requiring login name and password. They are forwarded to the ACA as “requester” information. The EAT provides a personalised GUI towards the logged end-user. For that reason, the EAT Manager looks into the end-user profile for former taken reservations, their characteristics, and the history belonging to the end-user.

The login mechanism is just for identification. At the moment exists no mechanism for the authentication of each user. The EAT can not prove the authentication of the logged in end-user. The login name and password are sent in plain text and they are sent over an insecure access network. Therefore the login mechanism of AQUILA architecture is insecure.

The previous subchapter discusses a few solutions for Authentication like DIAMETER or RADIUS. By RADIUS transactions between the client and RADIUS server are authenticated through the use of a shared secret, which is never sent over the network. In addition, any user passwords are sent encrypted between the client and RADIUS server, to eliminate the possibility that someone snooping on an insecure network could determine a user's password. One of these solutions should be included within the AQUILA architecture. But it is not the aim of the AQUILA architecture to develop a new mechanism for the authentication.

4.5 Provisioning control loops

In the current Aquila RCL, AC decisions are taken independently at individual ingress and egress ERs based on assigned AC limits. AC limits control the traffic volume that is allowed to pass an ER. Traffic is admitted up to the AC limit regardless of its destination (ingress AC) resp. source (egress AC)[2]. As a result AC cannot control resource utilisation inside the network. Only provisioning takes care about resource utilisation of core routers. Provisioning uses traffic forecasts to estimate traffic volumes on all links. If real traffic differs from forecasts, then some links will be overloaded and QoS targets will not be met. To avoid these QoS problems provisioning control loops must be used to adapt AC limits, RP limits and WFQ weights to actual traffic demands.

[2] In case of p2p AC a flow will be admitted, if both ingress and egress AC accept it (TCL 1, 2, 3). In case of p2a AC ingress AC is the only one that is asked and has to accept (TCL 4).

Control loops consists of:

- measurements of traffic volume and distribution to egress ER
- recalculation of AC limits and WFQ parameters based on actual traffic measurements
- decision whether to use the new AC limits and WFQ parameters or not
- distribution of new values to ACAs and RCAs if necessary

Control loops:

- control resource utilisation inside of core networks and adapt AC limits and WFQ weights to avoid QoS problems
- shift resources between TCLs

The remaining part of this chapter is organised as follows. Section 2 investigates the need for provisioning control loops. Section 3 shows that provisioning control loops can be validated with a small number of routers in the 2nd trial. Section 4 is a first step towards an implementation and describes a possible provisioning control loop architecture.

4.5.1 Re-provisioning

Question:

Do we need control loops for provisioning?

Problem:

The current AQUILA AC can control resource utilisation at the network edge only (RPs extend this control to traffic collecting / distributing trees close to the network edge). Current AC is not able to control resource utilisation in the inner core of backbone networks. This can cause QoS problem through overloaded links / nodes.

Currently the provisioning procedure in D1301 calculates AC limits using traffic distribution matrices (normalised traffic matrices) that describe traffic forecasts. If during network operation real traffic departs from the traffic forecast that was used for provisioning, then QoS targets will not be met because of overloaded links.

Possible Solutions:

- Provisioning control loops that adapt AC limits to actual traffic conditions, which are difficult to be forecasted and change.
- Over-provisioning covering the worst case.

- Belief that traffic is sufficiently stable and accept a small probability for QoS problems. A small probability for QoS problems means that QoS problems will arrive with a small probability only. But when QoS problems arrived then they will stand for a long time. If a real large number of users is attracted by new information or services causing congestion, they will not be served within some seconds.

- MPLS can mitigate QoS problems, but is not a solution. The advantage of MPLS is that traffic streams can be forced to take different routes very easily in the case of congestion. This can prevent the need to change AC limits or WFQ weights. But re-routing is possible in the limits of available resources only. So resource needs still have to be estimated and resources allocation may not meet real needs. Re-routing of traffic around congested areas even can arise new congestion. Finally to re-route traffic via MPLS a similar control loop is needed to detect problems and free resources and to calculate better routes. So we end with a control loop anyway.

- QoS-routing can mitigate QoS problems, but is not a solution. QoS-routing can be used to re-route traffic around congested areas like MPLS. But re-routing is possible in the limits of available resources only.

So either our provisioning procedure in D1301 has to be changed to worst case provisioning or the AQUILA RCL has to be complemented with a control loop for re-provisioning.

Example 1

See Figure 4-15. Less then 10% of the ingress traffic of ER 1 is expected to leave the network via ER 2 in this example. Let AC limits at ER 1 and 3 limit ingress traffic to a maximum of 100 Mbps. In case of p2p AC with an egress AC limit of 100 Mbps at ER 2, at least 100 Mbps are needed at the link between ER 1 and ER 2 to carry worst case traffic load. 100 Mbps are needed at any link between ER 1 and ER 2 to carry worst-case traffic load. Taking the 4 egress links altogether 400 Mbps are needed to be equipped for each worst case (all traffic uses a single link).

In total the accumulated capacity of all 4 egress links of ER 1 has to be between 100 Mbps, which is the minimum, and 400 Mbps, which is the worst case. Even if an ISP installs twice the capacity needed according the forecast to be able to stand traffic fluctuations, this is still a factor two apart from the worst case. This example shows that the over-provisioning can be huge. In general worst case over-provisioning is hard to be estimated. It depends on ingress and egress AC limit and on network topology. A study similar to [13cor1016b] could yield reasonable numbers for example topologies.

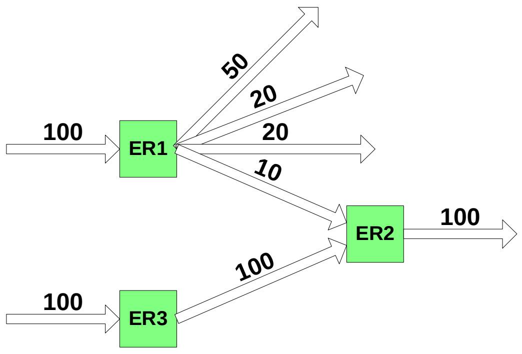

Figure 4-15: Fragment of a core network

Numbers show link capacities in Mbps which are needed for a certain traffic class according traffic forecast. Less than 10% of ingress traffic of ER 1 is expected take ER 2 as egress.

Example 2

Let AC be allowed to accept up to 1 Mbps at each ER in Figure 4-16.

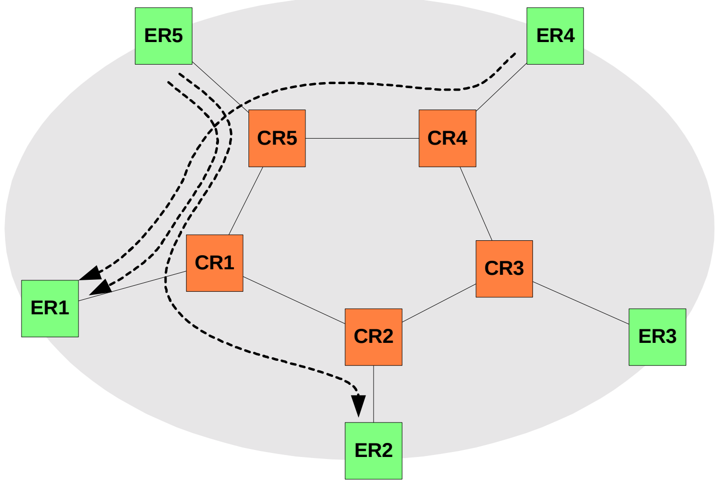

Figure 4-16: A network example

To indicate the network topology just a single edge router is connected to each core router. Equal routing link weights are assumed, so routing will use shortest hop counts. The link from CR 5 to CR 1 has to carry 3 traffic streams as indicated.

In case of equal traffic distribution (traffic spreads to all egress ER equally) 0,6 Mbps are needed between any pair of CR.

To cover the worst case 2 Mbps are needed between any pair of CR.

In this exemplary case over-provisioning ratio is 3,3 (worst-case resource needs divided by optimised resource needs). Of course 0,6 Mbps gives no room for any traffic fluctuations. So an ISP will install more than the minimum capacity. And the worst case may be unlikely to happen. But there is a great amount of BW that can be saved with a control loop and again only a study similar to [13cor1016b] can yield reasonable numbers.

Example 3

In case of p2a AC (not egress AC) worst case egress traffic at any ER in Figure 4-16 is 5 Mbps. That makes worst case provisioning not acceptable.

Are there any reasons to assume that traffic forecast can be wrong or traffic will change?

There are a number of reasons that make traffic forecasts very difficult:

- changes in inter-domain routing outside the network of an ISP can redirect large traffic streams (modified or new SLAs, new ISPs)
- new applications
- new services
- no experience with QoS traffic in IP networks
- daytime dependent traffic streams

A remark on control loops:

An ISP must measure offered traffic and traffic distribution across its network in a way similar to control loops anyway to be able to manage QoS provisioning and network growth.

4.5.2 Validation of Provisioning Control Loops in 2nd Trial

Figure 4-17 shows that provisioning control loops can be validated with a small number of routers in the 2nd trial:

- 4 edge routers are used only
- numbers give initially reserved BW for a certain TCL in Mbps
- initially 50% of the traffic entering the network via ER 1 cross ER 2 and the remaining 50% run to ER 4
- 6 Mbps are reserved for each egress link of ER 1 to have some space to detect load changes without QoS problems

- after some time traffic split at ER 1 will be changed from 1:1 to 1:4 for example (growing number of flows towards destinations behind ER 4), if the provisioning control loop works well, then AC limits will be adjusted

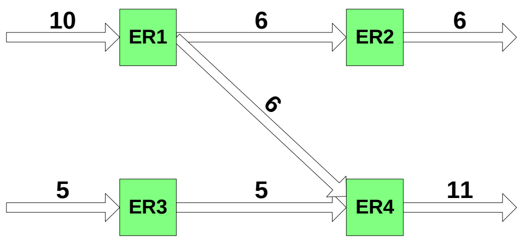

Figure 4-17: Example network configuration to test provisioning control loops in 2nd trial

4.5.3 Provisioning Control Loops

Goals

Adaptation of AC limits, RP limits and WFQ weights to real resource needs to avoid QoS problems and optimise resource utilisation.

Constraints

(I) Resources:
Available network resources are given and fixed. Given network resources are expressed in link capacities, i.e. BW. Provisioning cannot increase available BW. There is sufficient BW available in general. A partition of the available BW into individual shares for TCLs and ER that fits to resource demands has to be determined only. This is because AC limits and WFQ weights depend on BW partition and a mismatch can yields high blocking probabilities or QoS target violations. Thus a mismatch can result in lost revenues for the network operator and unsatisfied customers.

(II) Time Scale of Adaptation:
We assume that demand shifts that cause provisioning to adapt WFQ weights and AC limits occur infrequently only. May be resources have to be shifted between TCLs twice a day when most people start and end working, due to a shift in user behaviour. May be there are infrequent demand shifts that require re-provisioning in the order of days due to single events or long term trends.

(III) Static Inner Core:
There is an inner core that is static and BW assignment cannot be changed by a provisioning control loop there. Re-provisioning takes place in traffic collection and distribution areas near the network border only.

(IV) Low BW links:
Low BW links are not controlled by a provisioning control loop.

4.5.4 An Architecture of A Provisioning Control Loop

A possible architecture of a provisioning control loop in form of building blocks and their co-operation is described here. The algorithms that map resource demand measurements to the provisioning parameters AC limits, RP limits and WFQ weights and that will run inside these building blocks are described in D1302.

Figure 4-18 shows the proposed architecture for a provisioning control loop, which has the following major parts:

Forecast (yellow):

Provisioning is based on traffic forecasts. Each time provisioning is executed it will determine AC limits, RP limits and WFQ weights based on a new traffic forecast. In the first version conservative resource demand estimation functions are assumed and their output is used as the traffic forecast for the next provisioning period.

Traffic Measurement and Resource Demand Estimation (orange):

Measurement functions are connected to ACAs only, see bottom of Figure 4-18. BW used for reservations is measured per ER, if possible per ER pair. Further blocking frequencies are measured there. Measurements are based on AC functions, which have to provide information about blocking and amount of reserved BW. Based on these measurement values a resource demand traffic matrix which shows estimated BW demand for each ER pair and a blocking frequency matrix which shows estimated blocking probability per ER pair are generated. It is proposed to keep measurement and estimation distributed and not to collect all values in a central object (as proposed in [13cor1015a]). In this case demand and blocking matrix are dis-tributed.

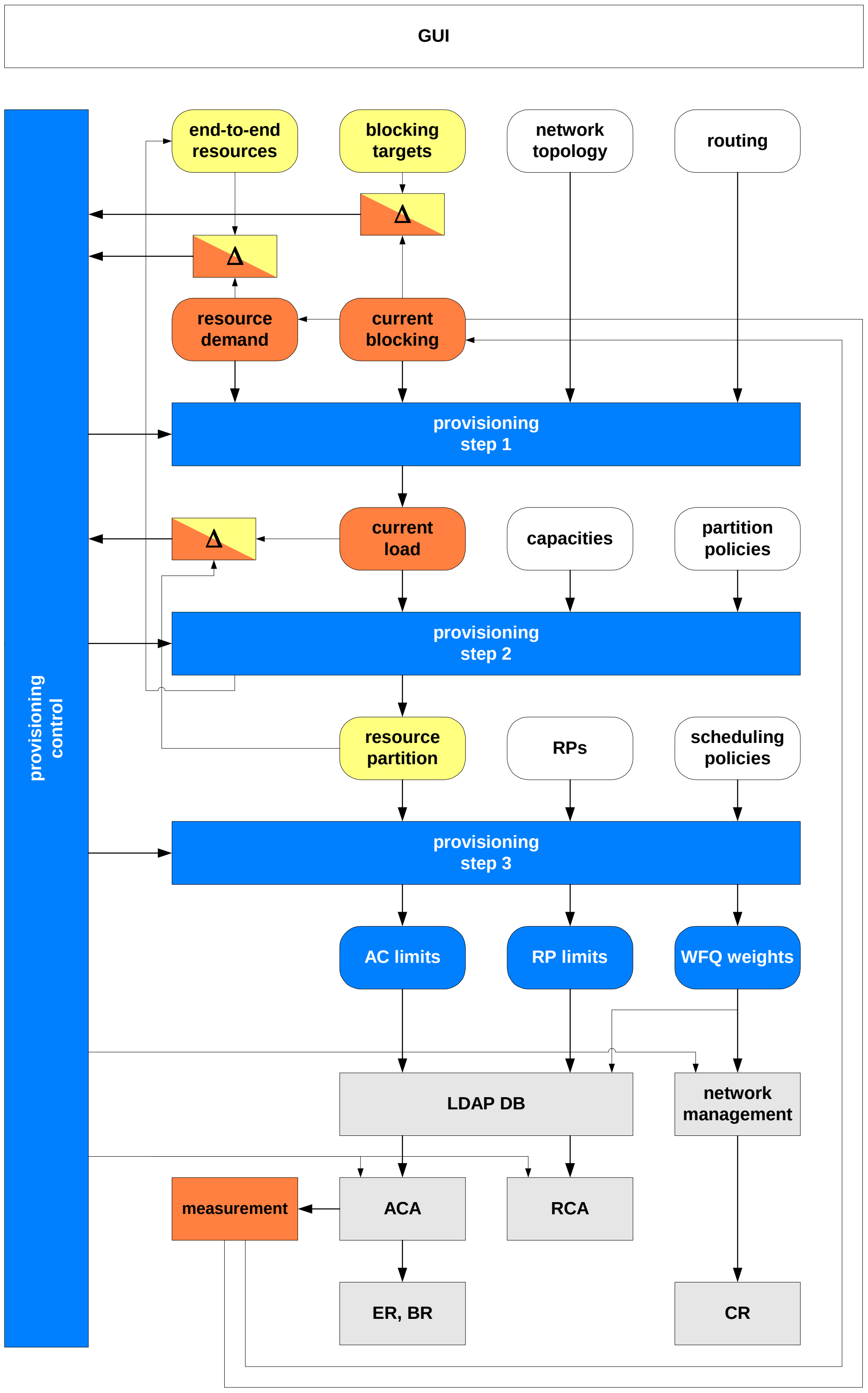

Figure 4-18: A control loop for re-provisioning

Provisioning (blue):

Provisioning has to calculate AC limits, RP limits and WFQ weights. Reconfiguration of these values should be done as infrequently as possible. Estimated resource demands and blocking frequencies are compared with forecasts in the boxes named 'Δ' in Figure 4-18 to determine if re-provisioning is required. If estimates exceed forecasted values which serves as thresholds, provisioning will be executed.

If demand and blocking estimations are distributed, thresholds should be distributed as well and triggering event will be sent to a central control object only. In this case provisioning can run after collection of distributed estimates only.

The final provisioning step is to distribute the new AC limits, RP limits and WFQ weights to the appropriate components and to make them work.

User Interface:

Of course a user interface is needed for a number of reasons:

- to feed the control loop with initial values (traffic forecasts, network topology, capacity, routing and provisioning policies)
- to control adaptations (display of need of re-provisioning, changes in traffic patterns, proposed new values)
- to control the control loop (measurement and estimation processes)

4.5.4.1 Building Blocks

'measurement'

Measurements are based on data provided by AC through an interface as proposed in [13sag1027a]. BW usage through reservations and blocking frequencies are measured per ER and ER pair if state based AC is used (see D1302). Measurements run on ingress as well as on egress AC.

To measure BW usage per ER pair a mapping of destination IP addresses to egress ER is needed. The function that yields the egress EAT can be reused here.

Measurement values are feed into an estimation process, that estimates BW demand, to eliminate short term load deterioration.

In a similar way blocking frequencies are estimated based on rejection information provided by AC.

'resource demand'

This is a conceptual building block that need not to be implemented. It holds the result of the measurement building block, that estimates BW demand based on information provided by AC.

'current blocking'

This is a conceptual building block that need not to be implemented. It holds the result of the measurement building block, that estimates blocking frequency based on information provided by AC.

'end-to-end resources'

This is a conceptual building block that need not to be implemented. It holds the available BW per TCL and ER pair. These are forecasts that serve as thresholds to trigger re-provisioning.

'blocking targets'

This is a conceptual building block that need not to be implemented. It holds the blocking probability targets. These are forecasts that serve as thresholds to trigger re-provisioning.

'Δ'

Compares estimated values with forecasts which serve as thresholds. Issue a notification if a threshold is exceeded.

'provisioning'

Implementation of provisioning procedure as proposed in D1302 based on collected demand and blocking estimates.

Reports proposed changes and reasons via the '*GUI*' to an administrator. If the administrator agrees, new values are stored in LDAP data base and ACAs, RCAs and management functions controlling core routers are triggered to further distribute the new values and to make them work.

It is proposed to split provisioning in 3 consecutive steps and an additional control block. ‘current load’ of each link and ‘resource partition’ to TCLs are major results that have to be calculated on the way to ‘AC limits’, ‘RP limits’ and ‘WFQ weights’ which are the final outcome.

Link ‘capacities’, ‘partition policies’, ‘RPs’ and ‘scheduling policies’ are further inputs used to calculate new provisioning parameter.

‘GUI’

A simple file interface can be used to input first traffic forecasts, network topology, available resources, routing and provisioning policies which will be static in the trial implementation of the RCL.

A first solution to track traffic changes, activity and provisioning results is to write them into a log file.

Connections between the ‘GUI’ and other building blocks are not shown in Figure 4-18.

Part III: Design and specification

5 Coarse design of system components

This chapter provides a coarse design specification as a starting point for the design and implementation of the system components. The AQUILA architecture is made up of five major components. Three of them were already in place in the first trial architecture:

- Admission control agent
- Resource control agent
- End-user application toolkit

The following two components are new:

- Inter-domain resource allocation
- QMTool

The following subchapters are arranged to reflect these components. For each component, the implemented functions and their interaction with the first trial architecture are roughly specified.

5.1 Admission control agent

5.1.1 MBAC

To support Admission Control Agent (ACA) in performing admission control based on traffic measurements, a new component called Measurement Based Admission Control (MBAC) is introduced. The following design model assumes that MBAC is an integral part of the ACA agent. The MBAC does not replace the declaration admission control methods developed for the first trial instead it extends the capabilities of AQUILA by providing additional admission mechanisms. It is assumed that the network operator will have to possibility in choosing the type of admission control algorithm on per traffic class and per edge device basis.

The following figure shows the MBAC logical structure as an integral part of the Admission Control Agent (ACA). The presented approach is based on router monitoring, where the aggregate traffic of existing reservations in the edge devices is monitored. The MBAC consist of two types of objects the MbacMonitor that is responsible for retrieving statistics data from the router and MbacProcessor that implements the AC algorithm.

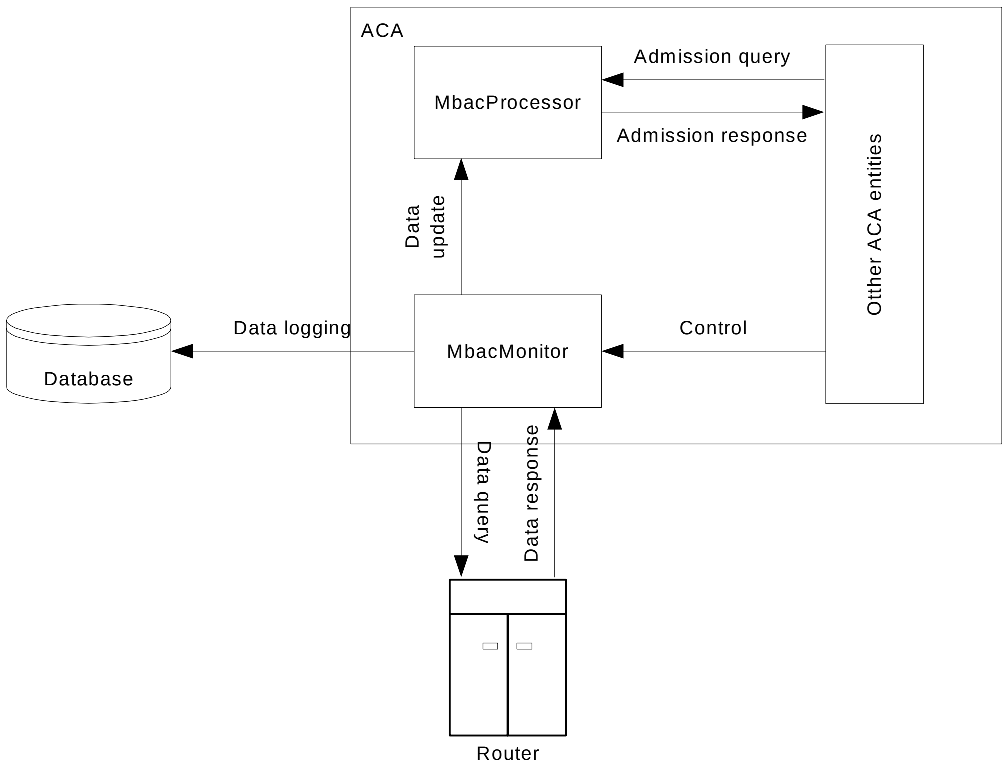

Figure 5-1: Logical structure of MBAC

The ***MbacMonitor*** monitors the Edge Device. It polls the router statistics at predefined rate and writes obtained data into the MbacProcessor entity(ies). To optimise the router polling mechanism one MbacMonitor is used per Edge Device to poll data for all traffic classes by single request. Therefore the polling interval for each TCL measurement AC algorithm is assumed to be the same.

In addition the monitoring data can be saved to the measurement database or logged to the file for evaluation of its performance. The MBAC monitor can be derived from the router QoS monitoring tool developed by ELI.

The ***MbacProcessor*** implements the admission control algorithm. It is periodically updated by the MbacMonitor with new measurement data. The MbacProcessor stores received data in his internal structures (for further processing) in a way specific to the admission control algorithm it implements. The MbacProcessor provides interface to other ACA entities that allow them to make admission queries. The ACA entity responsible for processing reservation request decides whether to admit new flow to the network, based on the response he gathered from the MbacProcessor entity.

5.1.1.1 The MBAC design model

The design model of MBAC is depicted on Figure 5-2. For each edge device one **MbacMonitorImpl** object is created if measurement-based admission control algorithm is to be used at least for one traffic class. The **MbacMonitorImpl** creates descendants of **MbacProcessorImpl** class for each TCL that will use the MBAC. The **MbacMonitorImpl** obtains its parameters (polling interval, information for which TCL to use MBAC) form LDAP database. The **MbacMonitorImpl** implements **MbacMonitor** interface that allows other ACA entities to start or stop the measurement procedure for given class, obtain reference to the MbacProcessor(s) or obtain information about the current status of the measurement procedure. In case the **MbacMonitorImpl** object is not able to obtain the router statistics it should deactivate the MbacProcessors.

The descendants of **MbacProcessorImpl** class implements the AC algorithms for given traffic class (one class for each TCL is defined). The **MbacProcessorImpl** class implements interface MbacProcessor that allows other ACA entities to make the admission control queries using AC method. This method should receive as parameters the traffic specification of current request as well as traffic specifications for all previously accepted flows. The method isActive indicates if the MbacProcessor is working correctly and can answer admission queries.

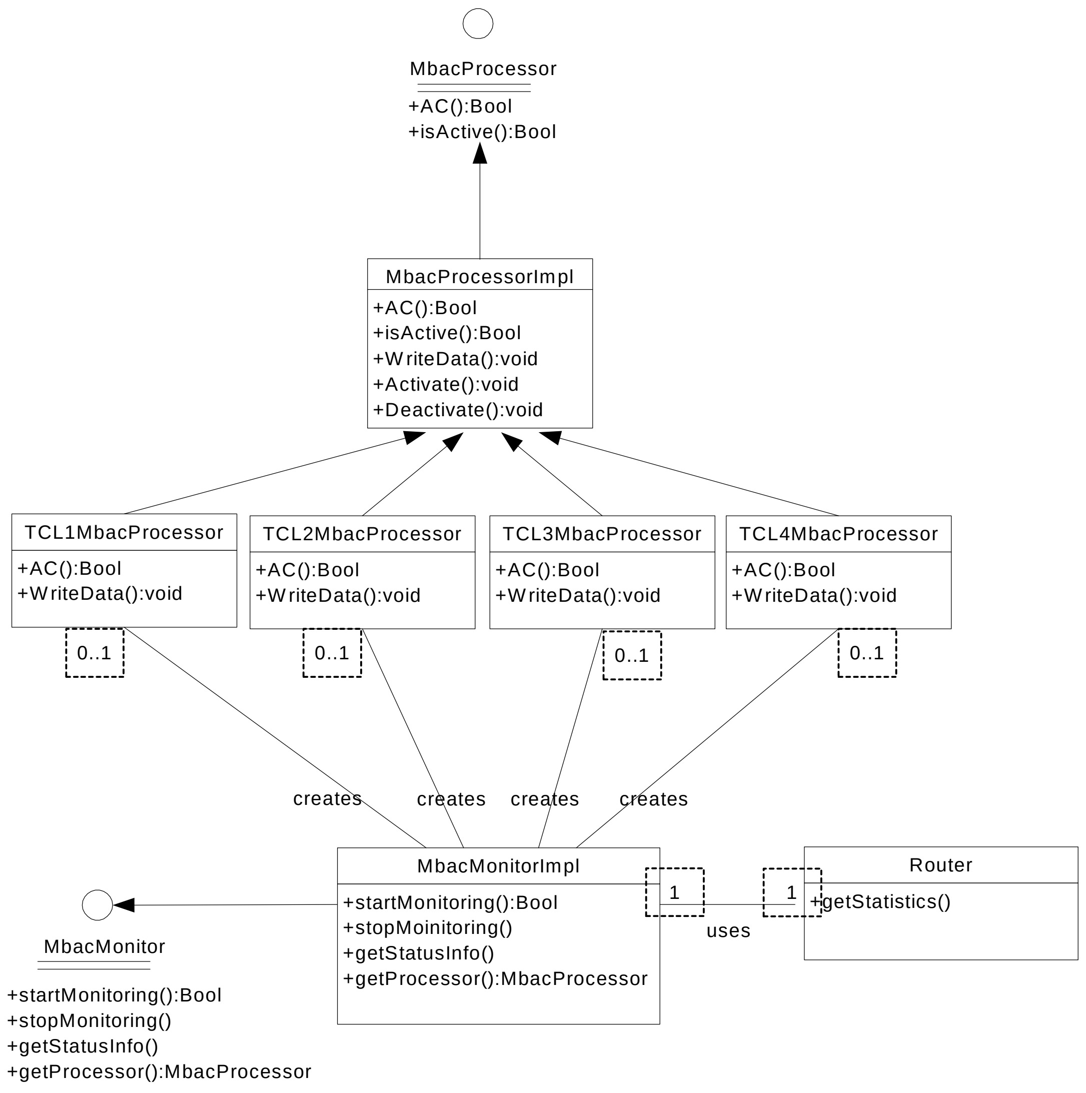

Figure 5-2: Coarse design of MBAC

5.1.2 Reservation groups

The proposal for Reservation Groups in chapter 0 introduces some changes in the ACA structure. The specification of these changes corresponds to this chapter.

5.1.2.1 Main structural changes

In order to understand better the changes introduced by reservation groups, the ACA structure diagram, presented in [D1201] chapter 4.2, is modified in Figure 5-3

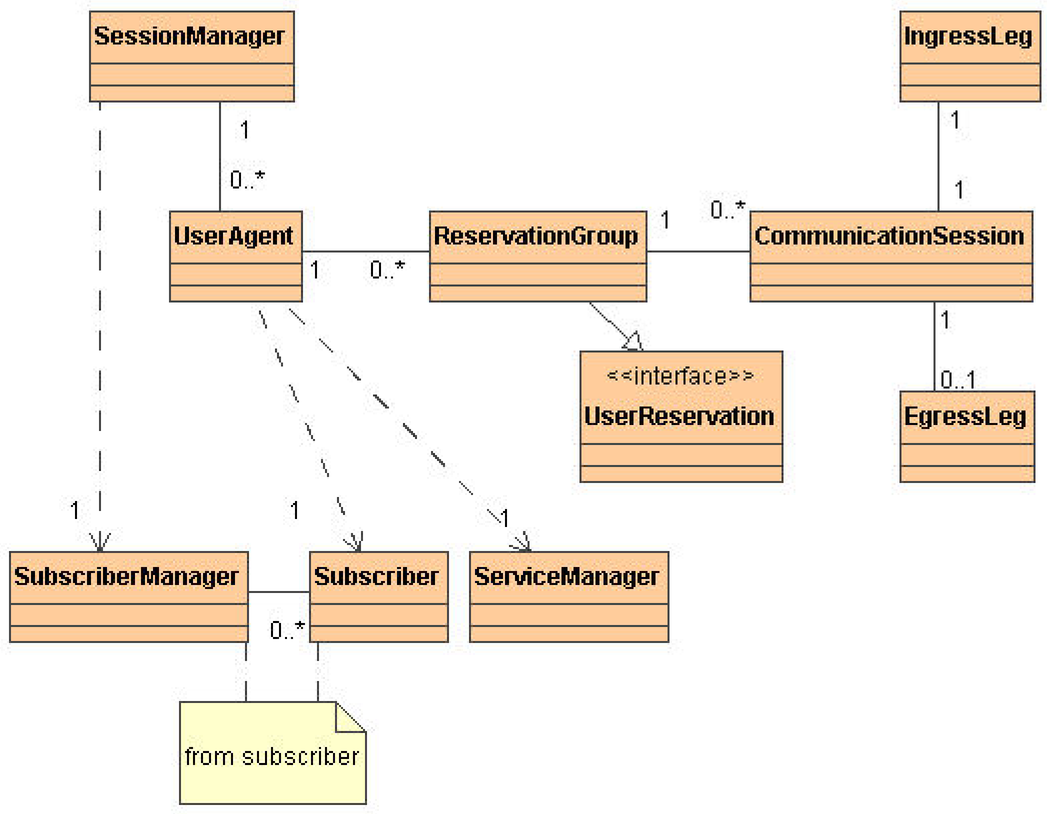

Figure 5-3: Structure of the ACA with Reservation Group

The ***SessionManager*** interface provides the login method. The EAT uses this method in order to try to log in the user. Each time an EAT performs a successful login a ***UserAgent*** is instantiated, which represents an active, authenticated end-user connected to this EAT. The ***UserAgent*** provides the interface used by the EAT to establish reservations.

Before Reservation Groups the ***UserReservation*** interface was implemented by the ***CommunicationSession***, which represents a reservation. With the new structure the ***UserReservation*** interface is implemented by the ***ReservationGroup***. The EAT sends a vector of reservation requests to the ***ReservationGroup*** and it will create so many ***CommunicationSession*** as needed. If a single reservation is requested, the ***ReservationGroup*** will generate only one ***CommunicationSession***.

5.1.2.2 Processing of a Reservation Request

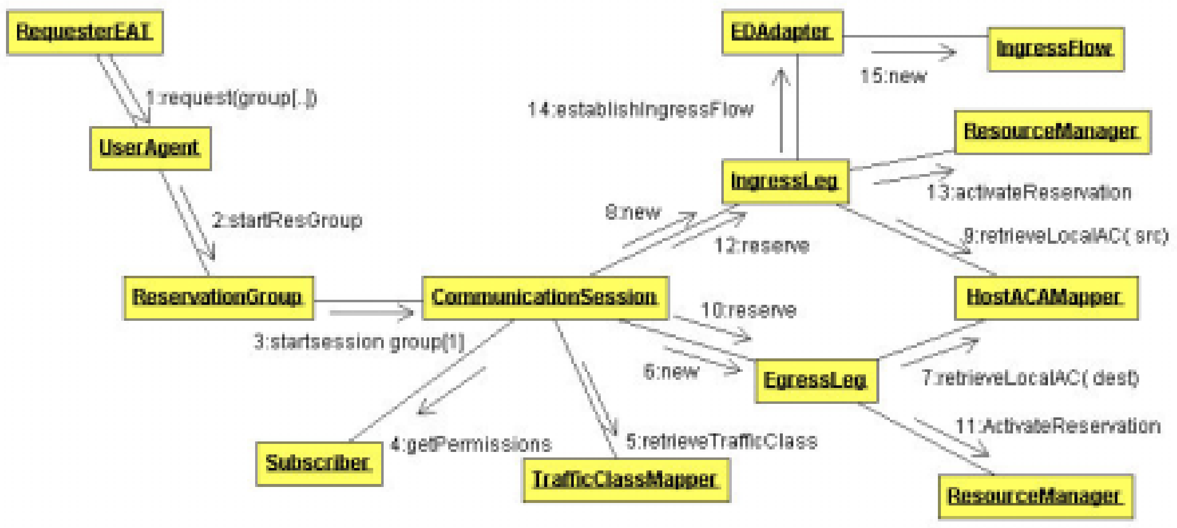

Figure 5-4: Processing of a Reservation Request

Figure 5-4 shows the processing of a single reservation request (that means, group has only one element). For a multiple reservation the ***ReservationGroup*** will create so many ***CommunicationSession*** as single reservations are requested. The different steps to process a reservation are:

1. A reservation request is initiated by an EAT.
2. The ***UserAgent*** representing this ACA creates a ***ReservationGroup***
3. The ***ReservationGroup*** creates a ***CommunicationSession***
4. The ***CommunicationSession*** checks whether the corresponding subscriber has the proper rights to do such a request for the requested network service.
5. The ***CommunicationSession*** checks if the traffic specification is allowed for the network service.
6. The ***EgressLeg*** is instantiated by the ***CommunicationSession*** (a p2p reservation is supposed).
7. The ***EgressLeg*** retrieves the responsible ***ACAResourceManager***, that has to execute the reservation in the destination.
8. The ***IngressLeg*** is instantiated by the ***CommunicationSession***.
9. The ***IngressLeg*** retrieves the responsible ***ACAResourceManager***, that has to execute the reservation in the source.

10. The ***CommunicationSession*** starts a reserve request in the ***EgressLeg*** . This request initiates an activation of earlier requested resources up to the network devices.

11. The ***EgressLeg*** activates the requested resources at the ***ACAResourceManager***, which is responsible for this reservation request.

12. The ***CommunicationSession*** starts a reserve request in the ***IngressLeg***. This request initiates an activation of earlier requested resources up to the network devices.

13. The ***IngressLeg*** activates the requested resources at the ***ACAResourceManager***, which is responsible for this reservation request.

14. The ***IngressLeg*** calls its ***EDAdapter*** to establish the flow, i.e. set up classifier/marker/policer state at the ingress ED.

15. The ***EDAdapter*** establishes classifier/marker/policer state, which is represented in the model by a new ***IngressFlow*** object.

5.1.2.3 Implementation model

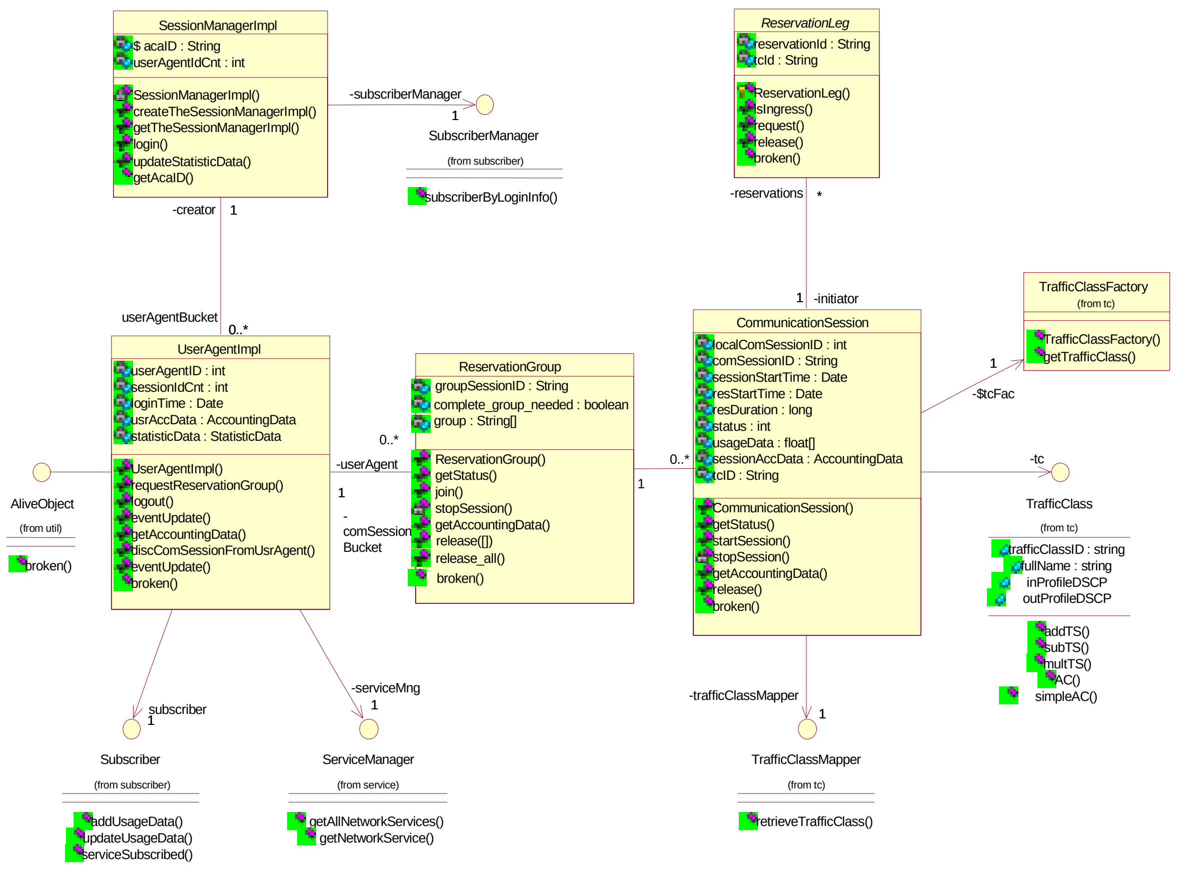

Figure 5-5: Implementation model ACA core

Figure 5-5 shows the modified implementation model of the ACA core. Only the main methods/attributes are shown. To this ACA core the ***ReservationGroup*** is added between the ***UserAgent*** and the ***CommunicationSession***. The ***ReservationGroup*** receives the reservation request with a ***group[]*** of reservations. A ***CommunicationSession*** is instantiated for every element of ***group[]***. All the information relative to the reservations has to be kept in the ***ReservationGroup***. This information has to be used for management operations, like releasing the established reservations when the parameter ***complete_group_needed*** is activated in the reservation request, and at least one reservation of the group could not be successful. The accounting data is also transmitted from the ***CommunicationSession*** to the ***UserAgent*** through the ***ReservationGroup***. The method ***join()*** offers the possibility to add a new reservation to the group and the method ***release([])*** offers the possibility to release only part of the reservation. When all the reservations of a Reservation Group have to be released, the method ***release_all()*** can be used.

5.1.3 Logging of usage data

The logging of Usage Data can be divided in the User part and the part for the ISP. The logged accounting data for the ISP is stored in a file. The ISP can post-process the file with its own accounting and billing software.

The End User retrieves the accounting data over the EAT. The End User can request the accounting data during the session or before he releases the session. At the EAT the method **getAccountingData** is part of the class **Reservation.**

For the Logging of **AccountingData** the ACA is extended with two components. The **AccountingDataLogger** collects at the manager ACA side the **AccountingData** of each **CommunicationSession.** The **UsageData** of each **CommunicationSession** is collected in the **UsageDataLogger** of the ingress ACA.

5.1.3.1 AccountingData

The class **AccountingData** is part of the **util** package. For the logging of accounting data, the **AccountingData** class must have the following attributes:

Session information:

- userID (int), sessionID (String)
- senderACA, receiverACA, managerACA (String)
- startReservationTime, endReservationTime (Date)
- durationReservation (float)
- startSessionTime, loginSessionTime (Date)
- terminateCause (String)

Reservation information

- TrafficClass, TrafficDespcription(trafficModel, trafficSpec) (package tc)

UsageData (from the ingress router):

- totalNumberOfPackets, conformedPackets, exceededPackets (long)
- totalNumberOfBytes, conformedBytes, exceededBytes (long)
- currentBurst (long)
- conformedRate, exceededRate (long)
- lastPacket (long)

Therefore the IDL description of the **AccountingData** has to be changed.

5.1.3.2 Retrieval of Accounting Data

The Accounting Data can be requested by three methods:

- The EAT requests for the **AccountingData** of an active Reservation (**getAccountingData)**.

- The EAT sends a **release** to the ACA. The **AccountingData** will be stored in a file before releasing the reservation. No **AccountingData** is sent back to the EAT.

- The EAT sends a **releaseWithAccounting** to the ACA. The **AccountingData** will be stored in a file and sent back to the EAT.

5.1.3.2.1 Retrieval from EAT (getAccountingData)

Figure 5-6 shows the retrieval of the **AccountingData** from the EAT. The request for the **AccountingData** is received by the manager ACA. The **CommunicationSession** sends the request to the ingress ACA. The **ACAResourceManager** of the ingress ACA is connected to the **IngressLeg** of the manager ACA. The **LocalResourceManager** forwards the request to the **IngressFlow** through the correspondent **SessionResource** and the **IngressFlow** queries the **UsageDataLogger.**

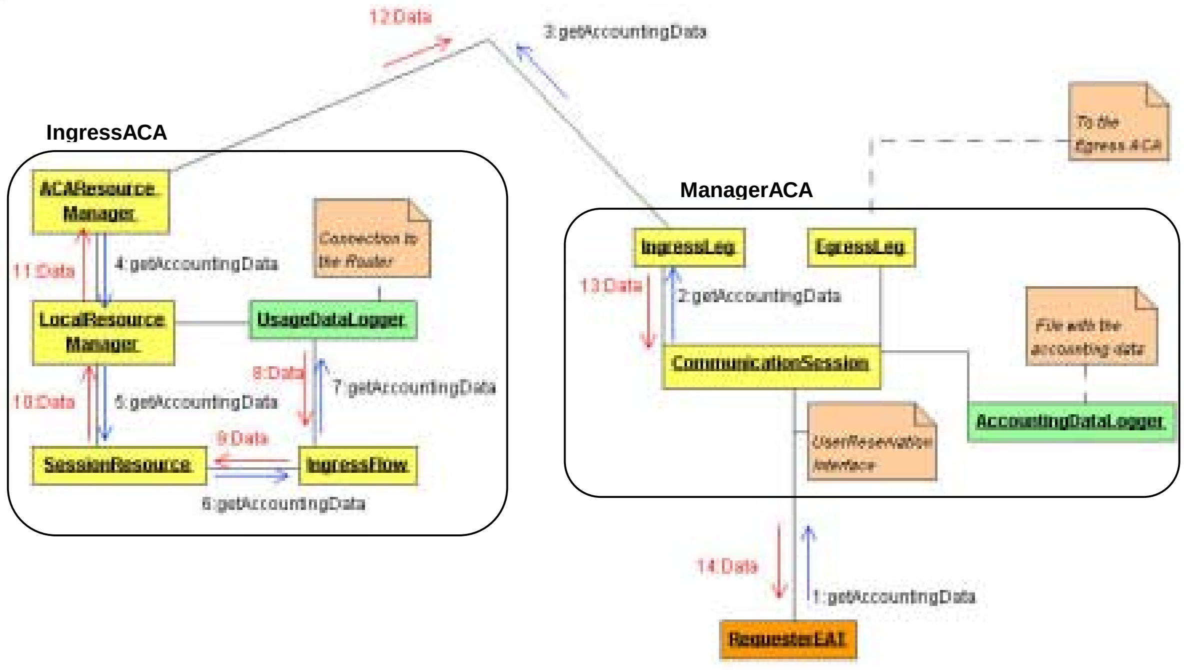

Figure 5-6: Retrieval from EAT

The **UsageDataLogger** polls the router regularly and stores the **UsageData** of all the flows. The information of all flows can be retrieved just by sending only one command to the router. The communication between the ACA and the Edge Device is a bottleneck of the AQUILA architecture and that is the reason for the **UsageDataLogger.** When a **getAccountingData** is received, the **UsageDataLogger** sends the **UsageData** for this reservation obtained in the last query. The fact that this information is not up-to-date is not important in this query. The user just want to get a “status” of the reservation.

The **IngressFlow** sends the **UsageData** back to the manager ACA. The **CommunicationSession** then adds the **SessionInformation** and the **ReservationInformation** to the **AccountingData.** The **AccountingData** will be sent back to the EAT. In this query no information is stored in the **AccountingDataFile**.

5.1.3.2.2 Release without AccountingData from EAT

The ACA offers the EAT two methods for the release. For the method illustrated in Figure 5-7 the EAT uses the method which gives no **AccountingData** back, this is the **release** method. The **CommunicationSession** of the manager ACA sends the **release** request to the ingress ACA.

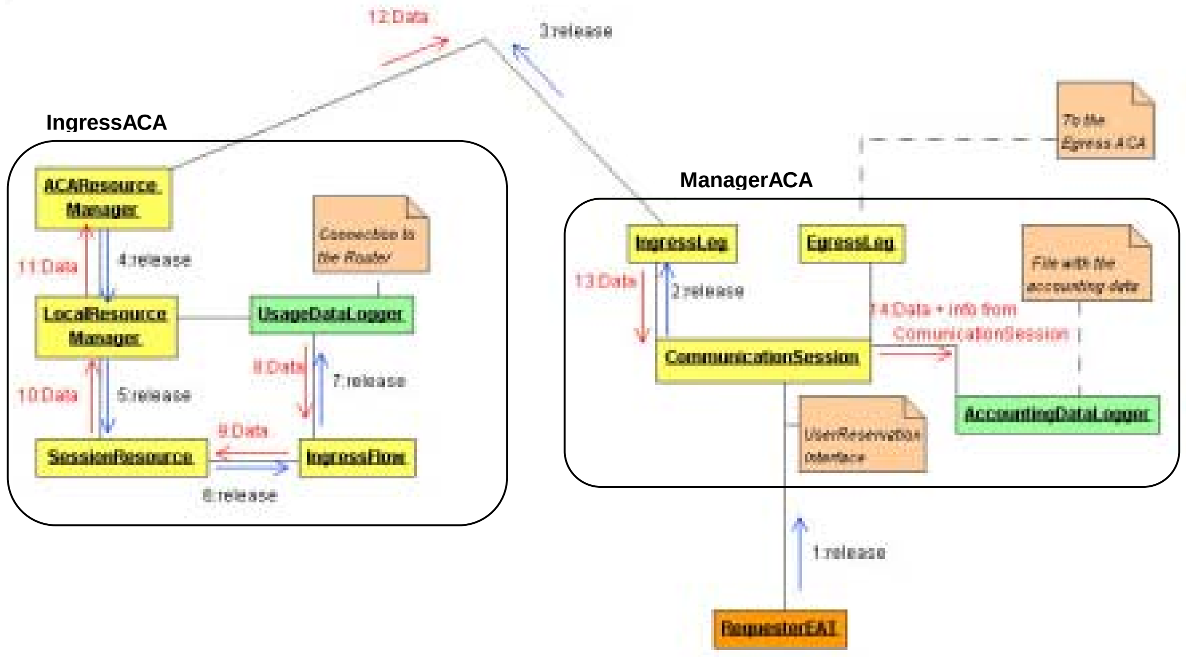

Figure 5-7: Release from EAT

The **LocalResourceManager** initialises the release of the **IngressFlow**. Before the **IngressFlow** quenches the configuration of the flow on the router the **UsageDataLogger** retrieves the **UsageData** of the flow. Immediately after the retrieval of the **AccountingData** the **IngressFlow** quenches the configuration of the flow. Thus, we can be sure that the user is not sending data “for free” after we retrieve the **AccountingData**.

The **IngressFlow** gives the **AccountingData** to the **LocalResourceManager**. The **UsageDataLogger** no longer stores the **AccountingData** of the flow. The **IngressFlow** and **SessionResource** are released and the ingress ACA has no information about the flow. The **CommunicationSession** of the manager ACA receives the **AccountingData** from the **IngressLeg** and adds the **SessionInformation** and the **ReservationInformation**.

The **AccountingDataLogger** gets the complete **AccountingData**, that is: the **UsageData,** the **SessionInformation** and the **ReservationInformation,** and stores it in a File. The stored data is for the post processing by the ISP. With this structure every ACA has an **Accounting-DataFile** where is stored all the accounting information for the reservations in which this ACA was in the Manager ACA role.

5.1.3.2.3 Release with AccountingData from EAT

The EAT can release the reservation with a second method: **releaseWithAccounting.** Then the EAT gets the **AccountingData** back from the manager ACA, see Figure 5-8 . The only difference we can appreciate on the ACA is in the **CommunicationSession.** When the **Com-municationSession** receives a **releaseWithAccounting** from the EAT sends a normal release to the Ingress ACA and waits until the **AccountingData** is received. Then **this Accounting-Data** will not only be sent to the **AccountingDataLogger** but also to the requester EAT. With this procedure we can be sure that the **AccountingData** the user is getting is the same that the ISP will use for the billing (coherence).

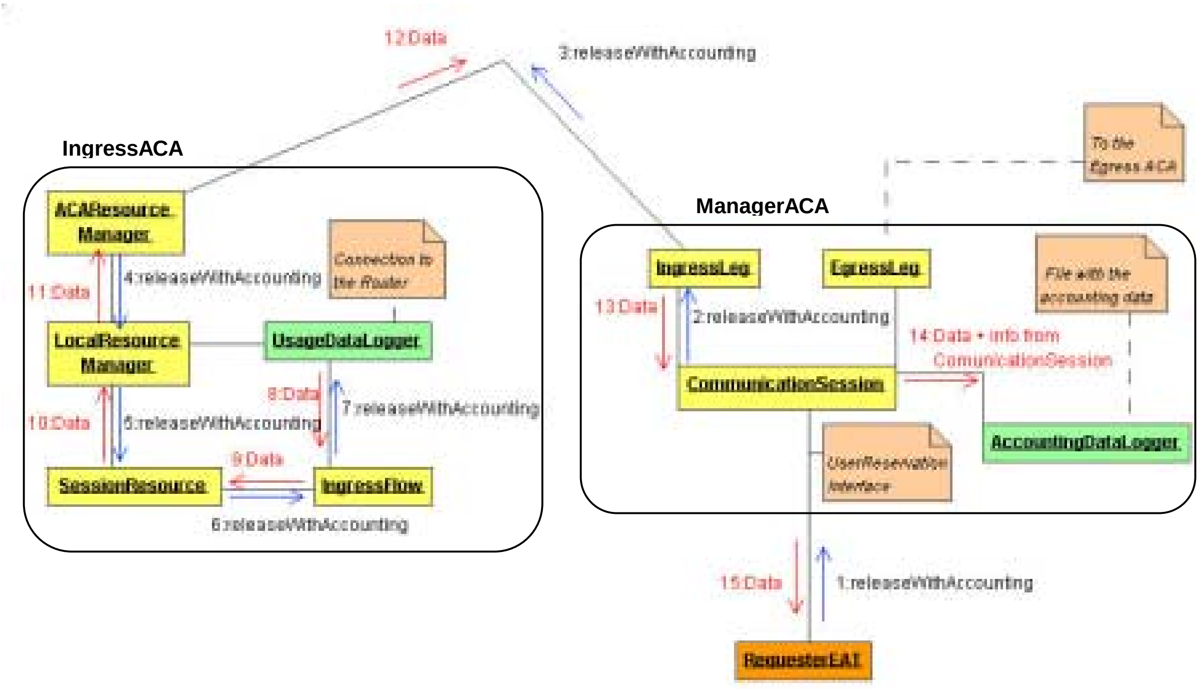

Figure 5-8: Release with AccountingData from EAT

5.1.3.3 AccountingDataLogger

The **UserReservation** builds the interface in the manager ACA to the requester EAT. For each **Reservation** the requester EAT collects the AccountingData and stores the Data in a Database.

The **UserReservation** builds the CORBA object towards the **CommunicationSession.** The **CommunicationSession** stores the SessionInformation for each session and the **Accounting-DataLogger** can retrieve the **sessionInfo** from the **CommunicationSession**. Figure 5-9 depicts the relation between the **AccountingDataLogger**.

The **CommunicationSession** initiates the **IngressLeg**. The **IngressLeg** builds the connection to the ingress ACA. The **CommunicationSession** can retrieve the UsageData of each **Reservation** from the ingress ACA over the **IngressLeg.**

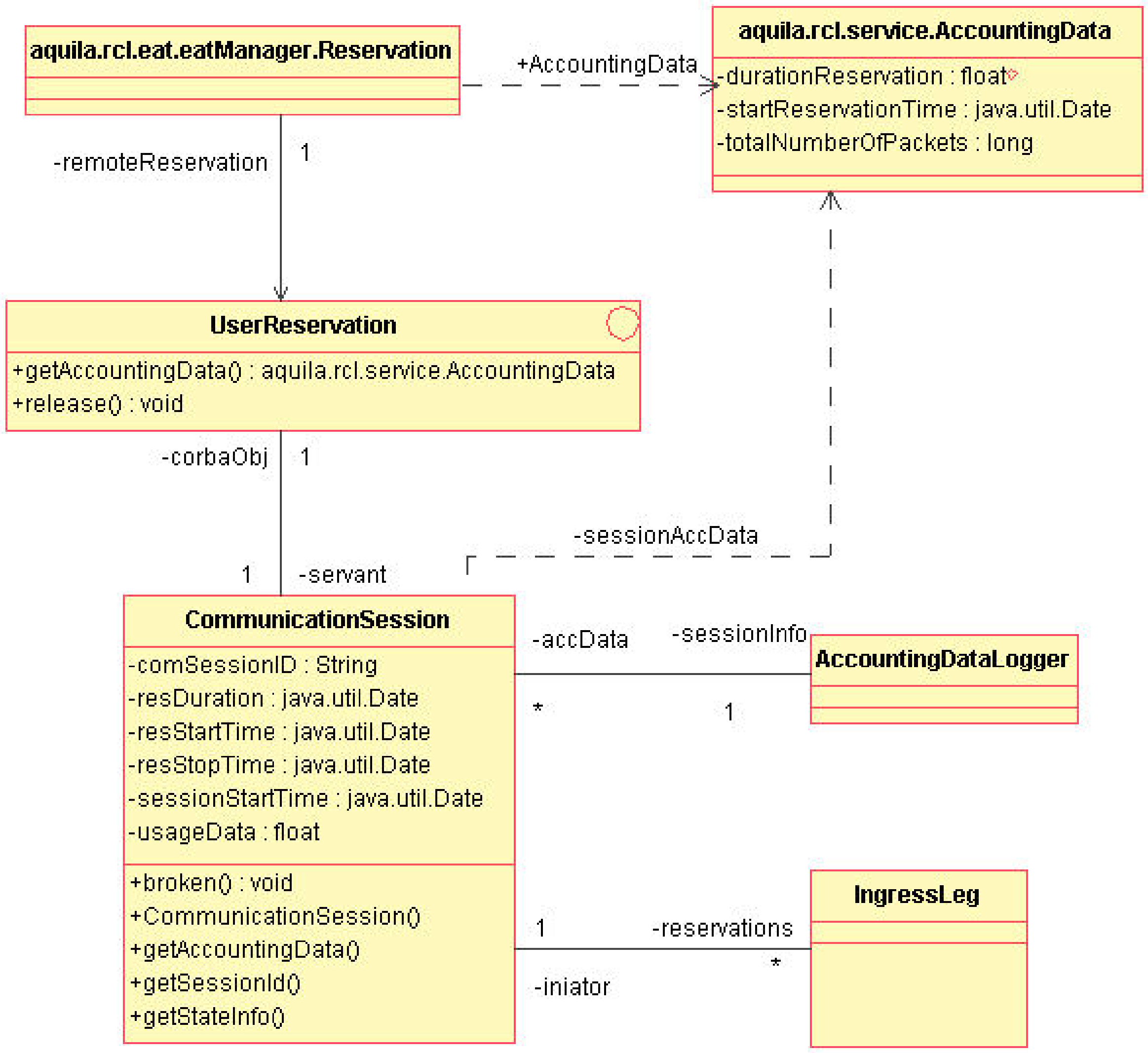
aquila.rcl.eat.eatManager.Reservation
+AccountingData
aquila.rcl.service.AccountingData
-durationReservation : float
-startReservationTime : java.util.Date
-totalNumberOfPackets : long
-remoteReservation
1
UserReservation
+getAccountingData() : aquila.rcl.service.AccountingData
+release() : void
-corbaObj
1
-sessionAccData
1
-servant
CommunicationSession
-comSessionID : String
-resDuration : java.util.Date
-resStartTime : java.util.Date
-resStopTime : java.util.Date
-sessionStartTime : java.util.Date
-usageData : float
+broken() : void
+CommunicationSession()
+getAccountingData()
+getSessionId()
+getStateInfo()
-accData
-sessionInfo
*
1
AccountingDataLogger
1
-reservations
*
-iniator
IngressLeg

Figure 5-9: AccountingDataLogger in the manager ACA

The **AccountingDataLogger** stores the **AccountingData** of each session in a file. The **AccountingDataLogger** exists the whole runtime of the ACA.

5.1.3.4 UsageDataLogger

The **UsageDataLogger** at the ingress ACA polls the UsageData of each **IngressFlow** from the EdgeRouter. The **LocalResourceManager** creates the **UsageDataLogger** during the initialisation. The UsageDataLogger is active the whole runtime of the ACA. The **UsageDataLogger** retrieves the **UsageData** for all Reservations.

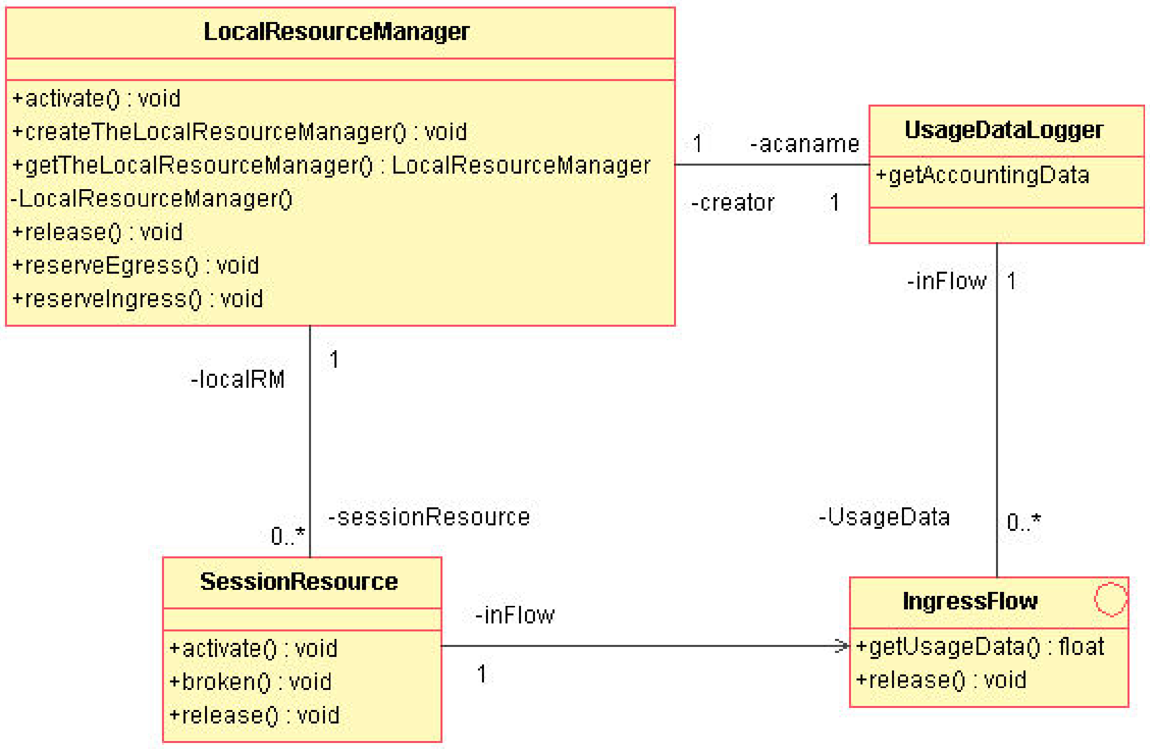

Figure 5-10: UsageDataLogger in the ingerss ACA

The **LocalResourceManager** (see Figure 5-10) controls every session with a **SessionResource**. The **SessionResource** creates an IngressFlow for each session on the router. The UsageData of each **IngressFlow** is based on the input policing of the flow.

The polling of the UsageData guaranties a saving of the AccountingData after a breakdown of the router. The UsageData is stored in the **UsageDataLogger** and can be retrieved from the **CommunicationSession** in the manager ACA.

5.2 Resource control agent

The following chapter is an update of the coarse design model that was included in D1201. It doesn’t describe the classes in detail since a more detailed description will be included in D2103.

The Resource Control Agent (RCA) is responsible for the distribution of the network resources to the ACAs and for performing admission control decisions upon reservation requests. To be more specific, the network topology is divided in a hierarchical manner to various levels of sub-networks: backbone network, sub-areas, subordinated sub-areas. The task of creating the resource pool tree is the responsibility of the network administrator.

The design of the RCA is mainly based on two key classes: the ResourcePool and the ResourceShare. The inner nodes of the tree are resource pools (**ResourcePool**), while the leaves (**ResourcePoolLeaf**) represent ACAs. Therefore, the resources assigned to each ResourcePoolLeaf are determined by the ResourcePools above it (i.e. its parent).

Finally, the RCA configures appropriately the ResourcePoolLeafs to be able to answer the requests for bandwidth allocations or de-allocations. The ResourcePoolLeafs perform the admission control decisions based on the bandwidth allocations determined by the resource pools and on the parameters specific to the adopted admission control mechanism.

The diagram illustrated in Figure 5-11 describes the above concepts using the UML notation.

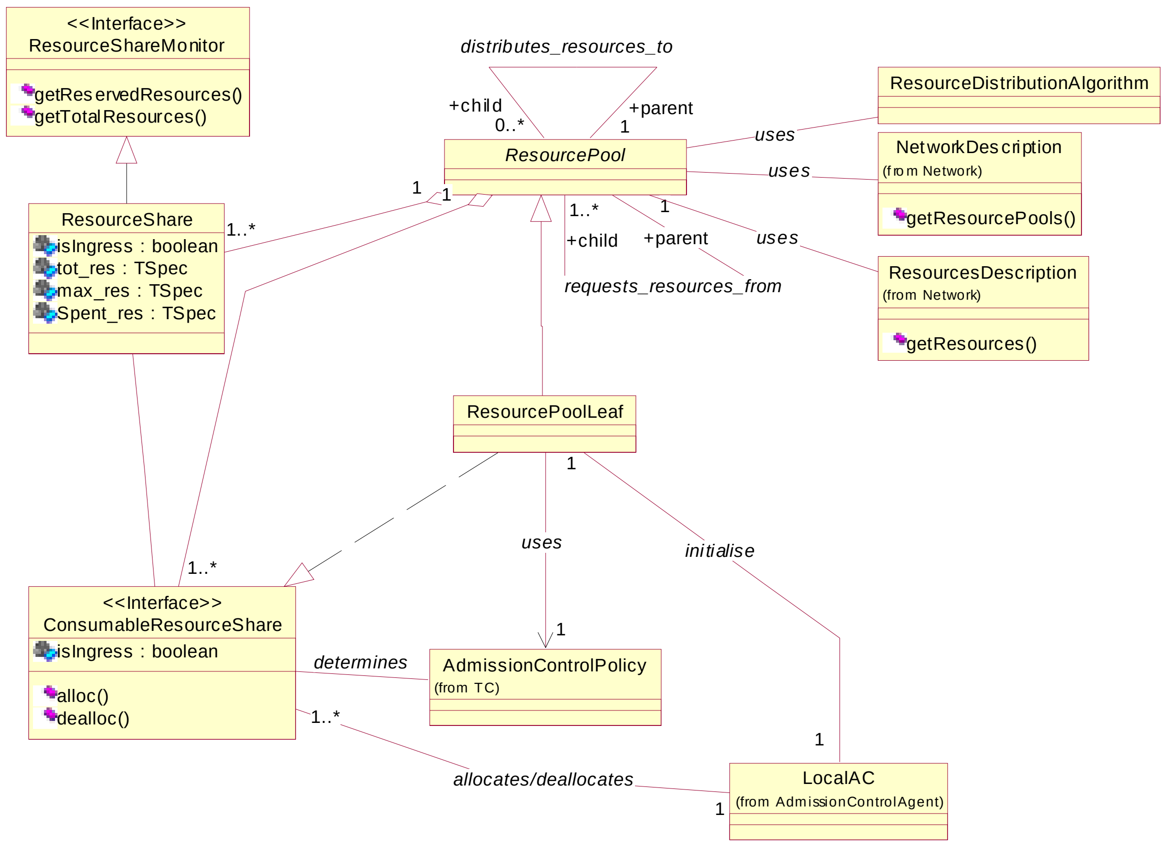

Figure 5-11: Coarse design model of Resource Control Agent

The parents ResourcePools distribute resources to the child ResourcePools and the leaves of the tree structure, which are initialised by the corresponding ACAs appropriately. Moreover, each ResourcePool uses a **ResourceDistributionAlgorithm** to distribute resources to the resource pools below it.

Each ResourcePool has been assigned some initial resources by the network administrator (initial configuration), which there are retrieved from LDAP, represented by the class ResourceShare. Each **ResourceShare** object refers to a specific traffic class and a specific direction (ingress or egress).The interface **ConsumableResourceShare** is implemented by the ResourceShare objects and represents, what an ACA sees from a ResourceShare object, i.e. ConsumableResourceShare will only be used at the interface between the ACAs and the RCA.

Whenever, an ACA "runs out" of resources the corresponding ResourcePoolLeaf, based on the **AdmissionControlPolicy** reconfigures the ACA accordingly. If needed the ResourcePoolLeaf might request more resources than the allocated ResourceShare from its parent resource pool and so on.

The ResourcePool has the following values for each traffic class and direction (ingress, egress):

- Max_res: the maximum amount of resources assignable to this pool
- Total_res: the total amount of resources assigned to this pool
- Spent_res: the amount of resources currently spent

If after a reservation the total resources are exceeded the ResourcePoolLeaf will request additional resources from its parent pool. If after a release the spent value is smaller than the low watermark the ResourcePoolLeaf will return resources to its parent pool. A request is immediately rejected if spent become greater than the Max_res.

In the above figure a new interface (**ResourceShareMonitor**) is also depicted. This interface is realised by the ResourceShare and used by the QMTool in order to monitor the state of each resource pool.

The algorithms used for resource distribution are described in deliverable [D1302].

5.3 Inter-domain resource control

Based on the analysis described in chapter 3.2, the following paragraphs depict a coarse design of the BGRP agent. Several issues regarding the design of the agent are highlighted and documented in class diagrams.

Many of the classes identified during analysis also appear here. However more classes are added, e.g. when a role attached to an association is actually more than just a reference. Also associations are refined, e.g. by changing the link to an association class to a "normal" association. Role names have been added on each navigable association.

However, there are still no attributes for the classes. This will be added in the design specification. Operations are given only, when they are used to provide accessibility.

5.3.1 BGRP agent

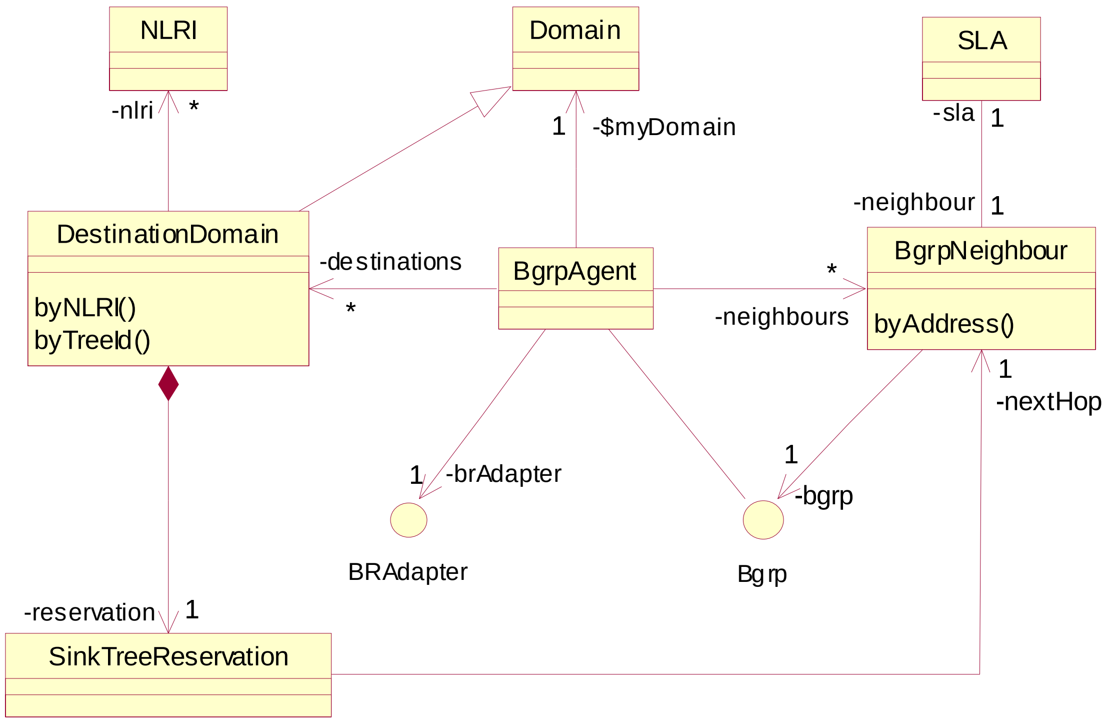

Figure 5-12: BGRP agent

The **BGRP Agent** is the central class. It implements the **BGRP** interface, which enables the **BGRP Agent** to receive BGRP messages. The main functions of message processing are implemented in the **BGRP Agent** class.

The **BGRP Agent** maintains two important tables:

- The list of **BGRP Neighbours** and
- the list of **Destination Domains**, towards which a reservation is currently active.

The **BGRP Neighbour** serves the following purposes:

- It is used to access the **BGRP** interface of a neighbour **BGRP Agent**;
- it stores and manages status information about the neighbour and
- it provides access to the **SLA** regarding the connection to this neighbour.

Destination Domain is a subclass of **Domain**, which keeps general information like the AS number. A **Destination Domain** object can be accessed either by performing a lookup in the NLRI table using an IP Address (to enable quiet grafting), or by using a sink tree id (for "standard" GRAFT processing). A **Destination Domain** object always contains a **Sink Tree Reservation**, which is used to access and process all information regarding the reservations along the sink tree to the given destination domain. It is a property of the sink tree that the path to the destination goes over a single next hop **BGRP Neighbour**.

The **BGRP Agent** has a reference to a **Domain** object describing its own domain.

The **Sink Tree Reservation** objects are responsible for all GRAFT, TEAR, UPDATE and ERROR processing associated with a sink tree. The **SLA** object connected to the associated **BGRP Neighbour** is used to check the conformance of the reservation with the **SLA**.

5.3.2 SLA

The BGRP agent uses SLA information to perform admission control on the domain level. SLA information is structured in the following way:

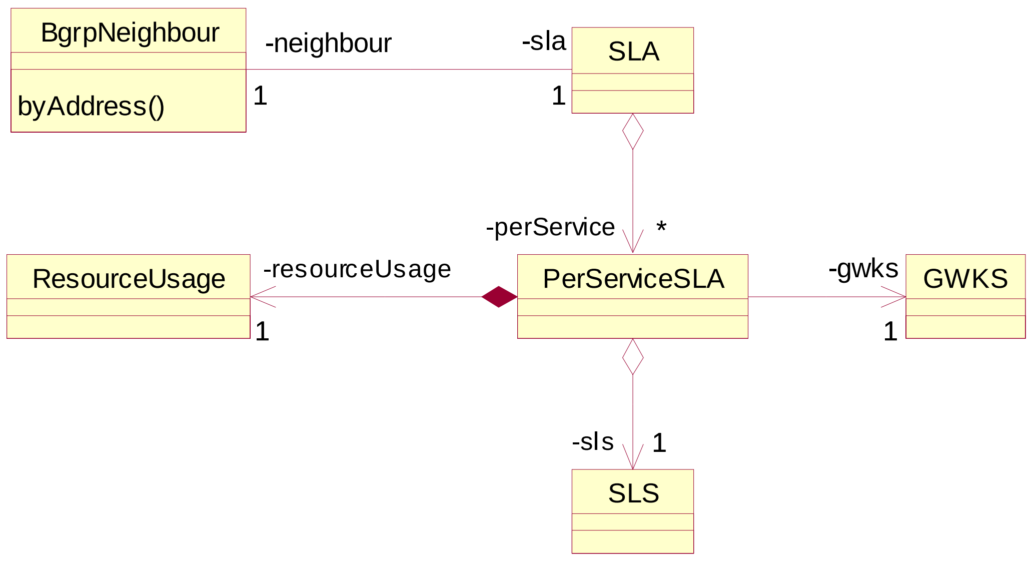

Figure 5-13: SLA information

An **SLA** may contain some general, service independent information and additionally a set of **Per Service SLAs**, each corresponding to a **GWKS**. The technical characteristics of a **Per Service SLA** are stored in an SLS. SLAs exist both for internal and external neighbours.

The **Resource Usage** records the extend, to which the SLA is used by the currently existing reservations. It manages both the currently requested (PROBE sent) and the actually reserved (GRAFT received) resources.

5.3.3 Reservations

The BGRP agent keeps information associated to the reservations linked to the sink trees to the destination domain. The following figure depicts the classes describing this information:

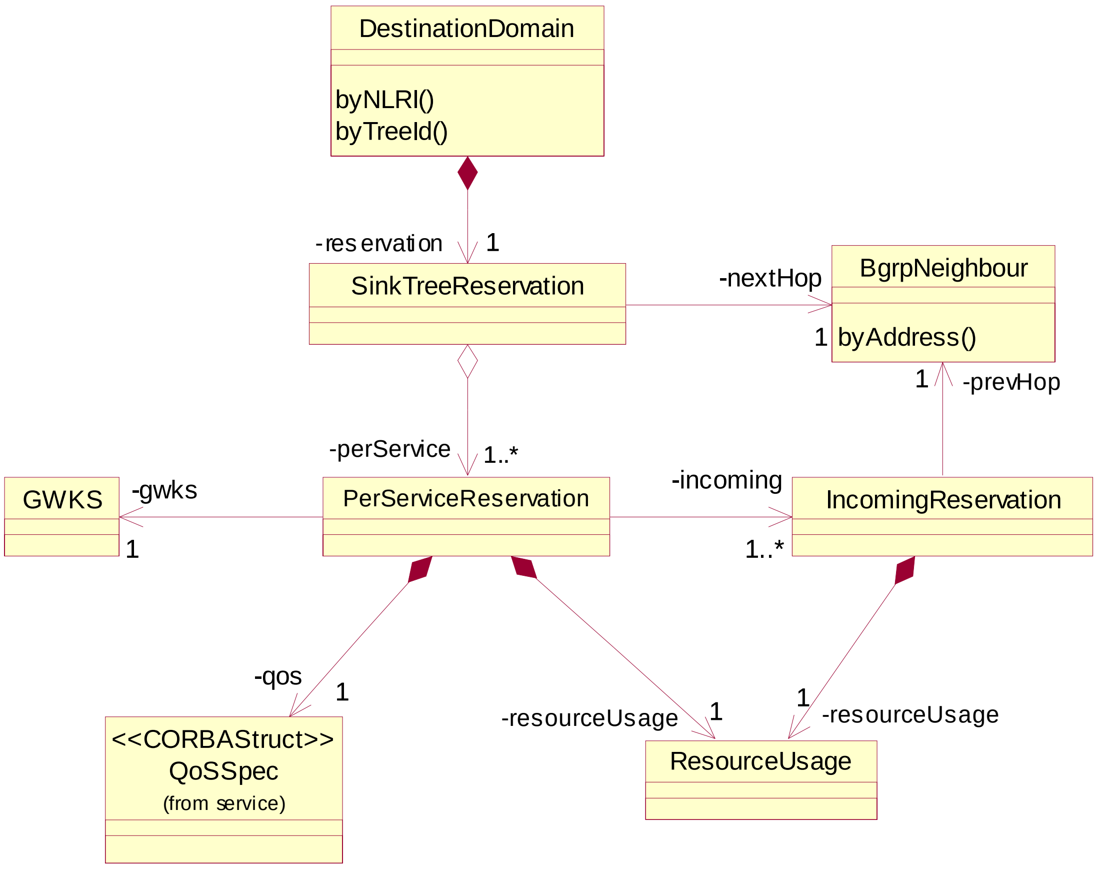

Figure 5-14: Reservation information

For each **Destination Domain** a **Sink Tree Reservation** exists. When no reservations are active to a given **Destination Domain**, then also the **Destination Domain** object is deleted.

It is a property of the BGP routing protocol that all packets for a given **Destination Domain** are forwarded over a single next hop BGP router, described by the **BGRP Neighbour** object.

Sink Tree Reservations are composed of one or more **Per Service Reservations**, each connected to a **GWKS**. **Per Service Reservations** store the bandwidth currently requested and reserved for that service and that sink tree in a **Resource Usage** object. They also store the accumulated **QoS Spec**, describing the treatment a packet will receive on the way to the ingress border router of the destination domain.

A **Per Service Reservation** has a list of all **Incoming Reservations**, which are aggregated at this BGRP agent and make up the (outgoing) **Per Service Reservation**. This list is necessary to process UPDATE messages between neighboured BGRP Agents. An **Incoming Reservation** stores the bandwidth currently requested and reserved for that service and that sink tree in a **Resource Usage** object.

A **Per Service Reservation** is deleted, if there are no more **Incoming Reservations**. A **Sink Tree Reservation** and its connected **Destination Domain** is deleted, if there are no more **Per Service Reservations**.

As stated above, **Resource Usage** is kept for the **Per Service Reservation** and for all **Incoming Reservations**. The **Resource Usage** class manages both the currently requested (PROBE sent) and the actually reserved (GRAFT received) resources. This enables a stateless processing of messages.

5.4 End-user application toolkit

The following chapter is based on and refines the coarse EAT design given in [D1201], [D2201]. This chapter does not repeat these information but gives additional and more detailed information on the EAT architecture and design, based on the new requirements for the EAT and the results/experiences of the first development and trial phase.

5.4.1 Overall architecture

The necessary modifications do not gravely influence the former defined overall architecture. As depicted in Figure 5-15, the refined EAT will consist of the same basic building blocks. The main difference is the absence of the former called "daemons"; the functionality of the RSVP Daemon is now covered by the RSVP Proxies, and no DiffServ Daemon/Proxy will exists.

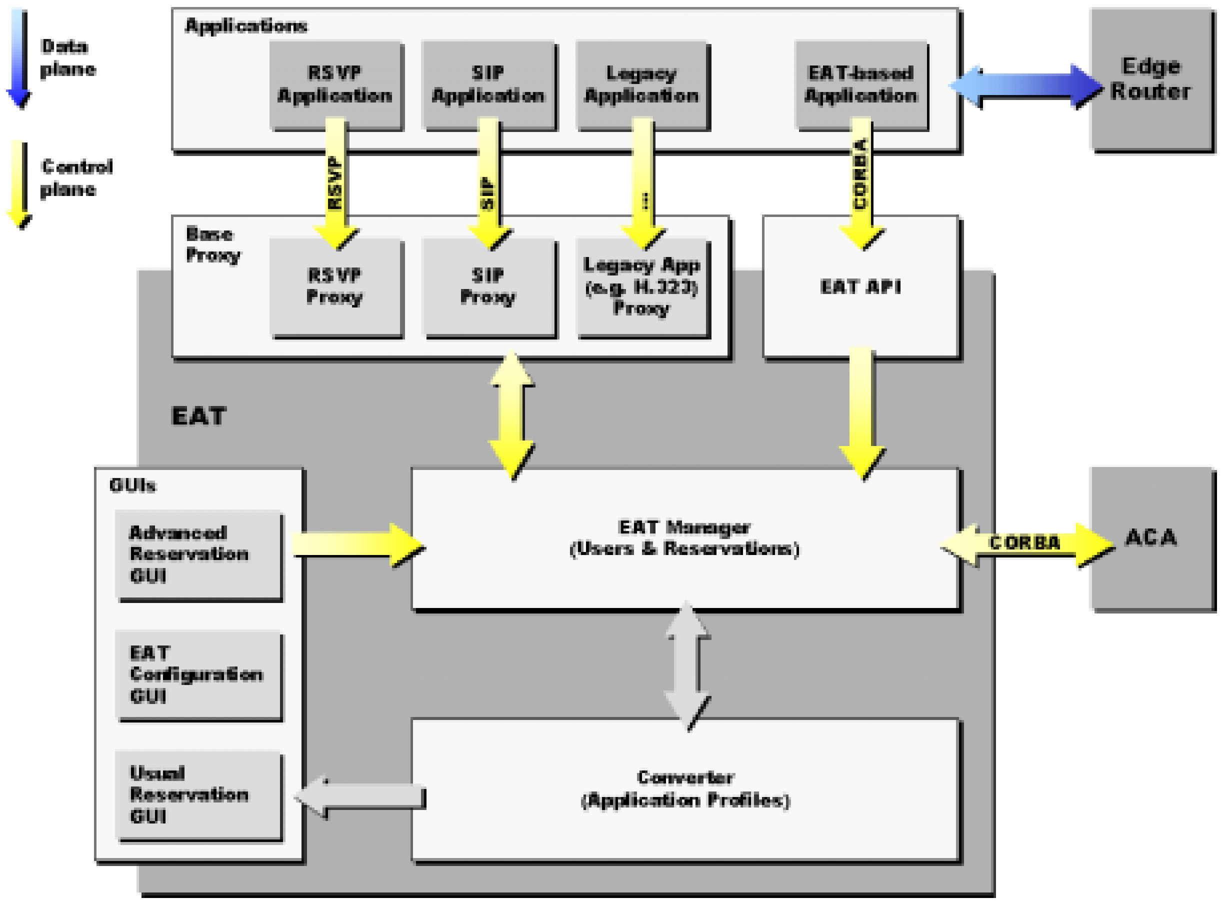

Figure 5-15: Block diagram of the overall EAT architecture

In order to realise the modular architecture above, the whole EAT package is divided into five sub-packages/components as shown below:

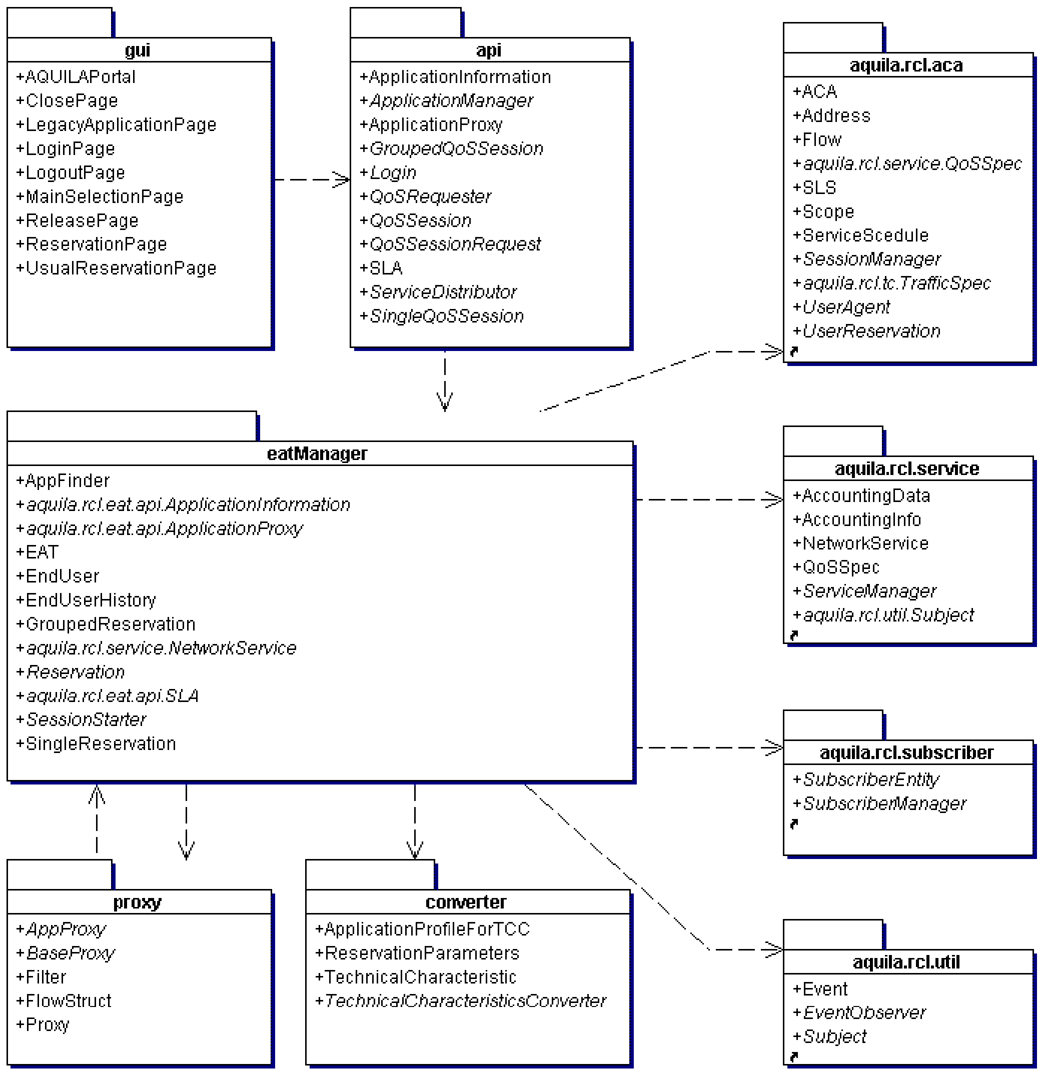

Figure 5-16: Package and dependencies within the EAT

The internal communication between the GUI and the API as well between the EAT Manager and the Proxy relies on CORBA. The interfaces are specified in IDL: api.idl, proxy.idl, and eatManager.idl. And although the Converter is directly used by the EAT Manager, an IDL interface specification exists in converter.idl.

In the following, more details of the single components are given.

5.4.2 EAT Manager

The EAT Manager, which is the central component of the EAT, responsible for user and reservation management, consists of the following classes (Figure 5-17):

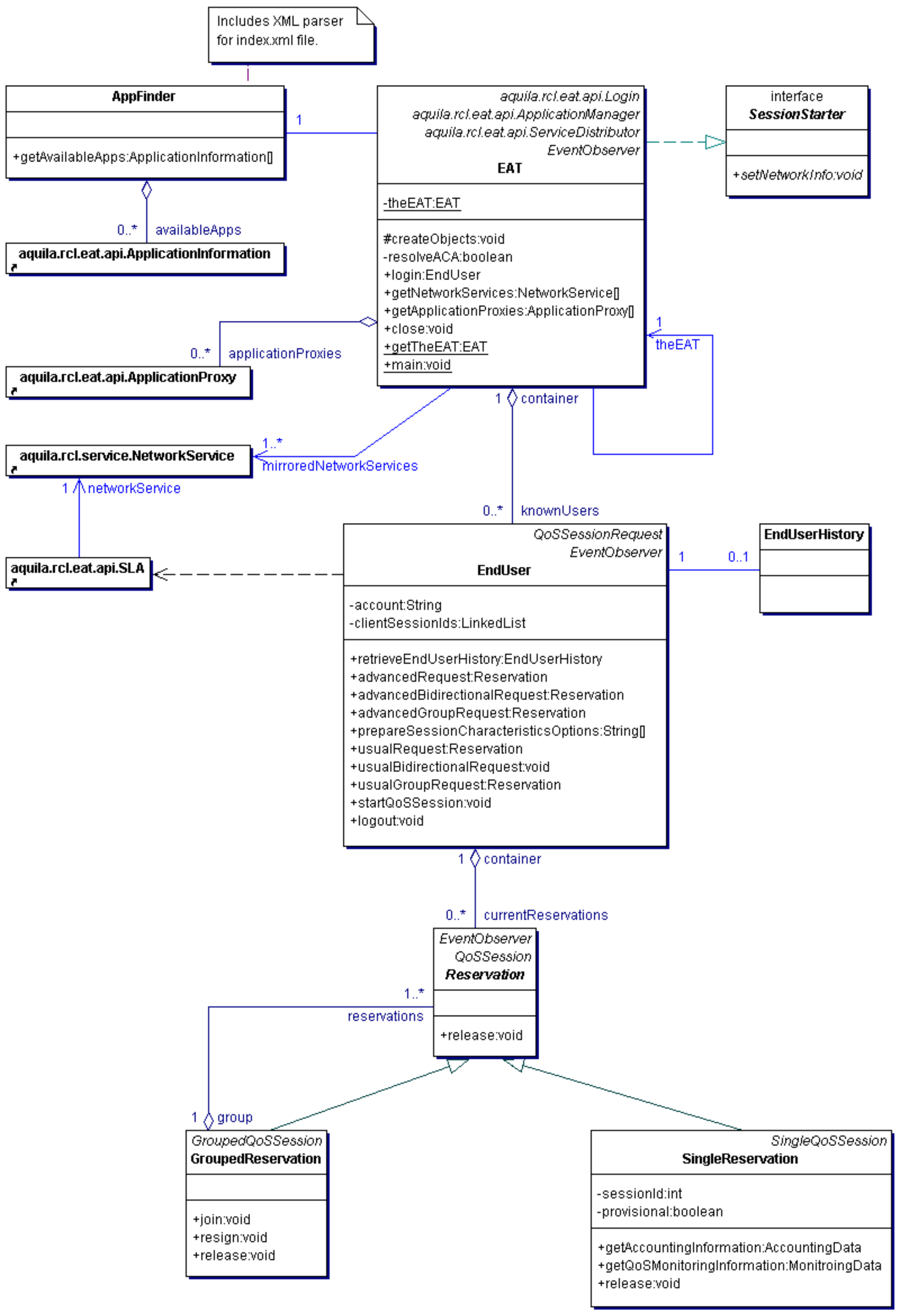
Includes XML parser for index.xml file.
AppFinder
+getAvailableApps:ApplicationInformation[]
1
0..*
availableApps
aquila.rcl.eat.api.ApplicationInformation
aquila.rcl.eat.api.Login
aquila.rcl.eat.api.ApplicationManager
aquila.rcl.eat.api.ServiceDistributor
EventObserver
EAT
-theEAT:EAT
#createObjects:void
-resolveACA:boolean
+login:EndUser
+getNetworkServices:NetworkService[]
+getApplicationProxies:ApplicationProxy[]
+close:void
+getTheEAT:EAT
+main:void
interface
SessionStarter
+setNetworkInfo:void
1
theEAT
0..*
applicationProxies
aquila.rcl.eat.api.ApplicationProxy
1 container
1..*
mirroredNetworkServices
aquila.rcl.service.NetworkService
1 networkService
aquila.rcl.eat.api.SLA
0..* knownUsers
QoSSessionRequest
EventObserver
EndUser
-account:String
-clientSessionIds:LinkedList
+retrieveEndUserHistory:EndUserHistory
+advancedRequest:Reservation
+advancedBidirectionalRequest:Reservation
+advancedGroupRequest:Reservation
+prepareSessionCharacteristicsOptions:String[]
+usualRequest:Reservation
+usualBidirectionalRequest:void
+usualGroupRequest:Reservation
+startQoSSession:void
+logout:void
1
0..1
EndUserHistory
1 container
0..* currentReservations
EventObserver
QoSSession
Reservation
+release:void
1..*
reservations
1 group
GroupedQoSSession
GroupedReservation
+join:void
+resign:void
+release:void
SingleQoSSession
SingleReservation
-sessionId:int
-provisional:boolean
+getAccountingInformation:AccountingData
+getQoSMonitoringInformation:MonitroingData
+release:void

Figure 5-17: The EAT Manager design model

Whereas EAT (singleton main class), EndUser (user management) and Reservation (reservation/QoS session management) were already defined in the previous version, the new classes are:

- AppFinder, to parse and hold information on installed application profiles,
- EndUserHistory (replaces EndUserProfile), to control history (e.g. former reservation) data of an end-user,
- GroupedReservation, to realise grouped reservation requests (see 3.3.2.1), and
- SingleReservation (former: Reservation which is now *abstract*), for single reservations like in the first trial.

It is important to mention that the EAT Manager *directly* implements the internal EAT API interfaces (see chapter 3.3.2.1). In other words, the API is the interface of the EAT Manager towards applications and GUIs (Figure 5-18).

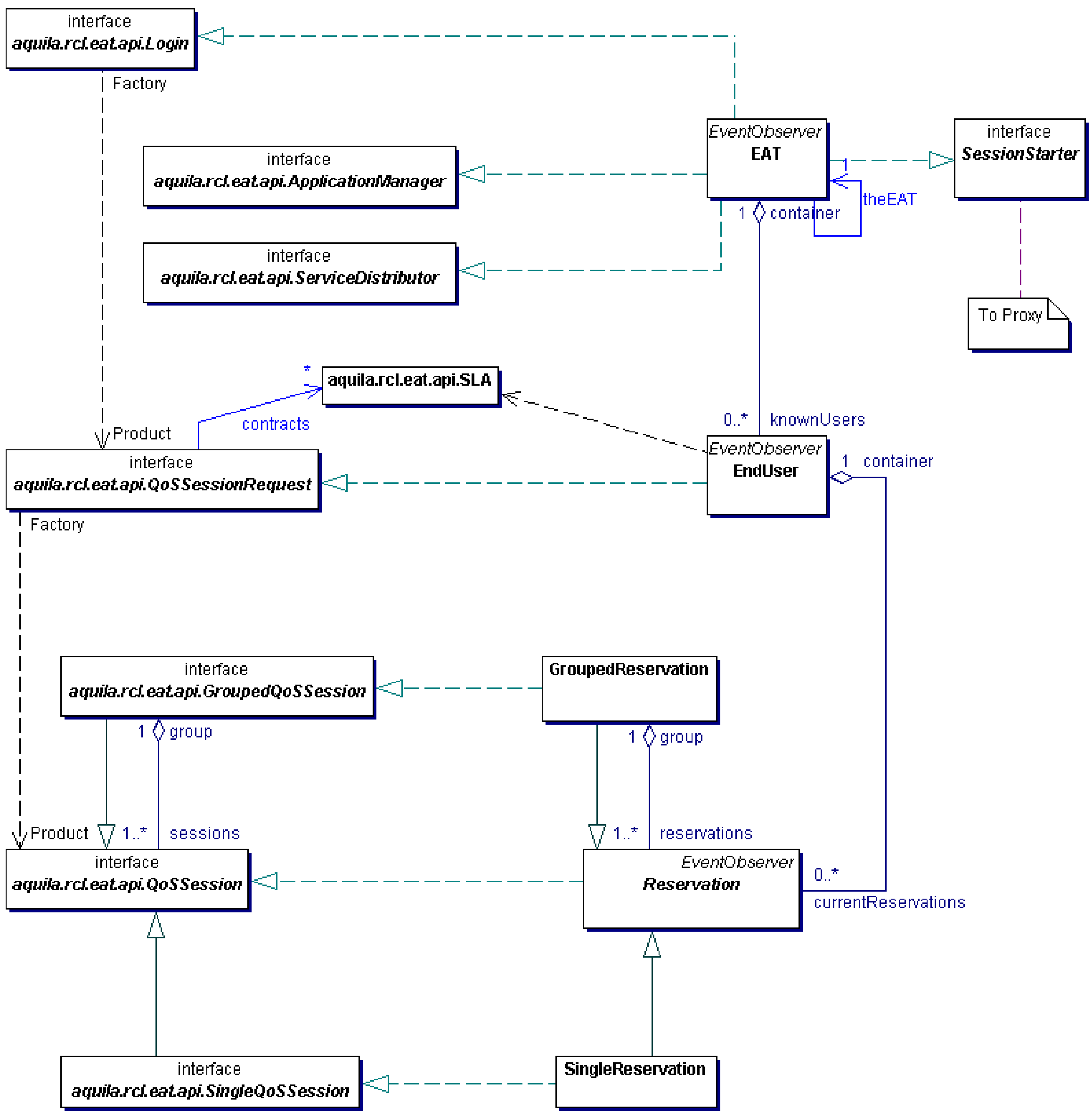

Figure 5-18: EAT Manager classes implementing the API interfaces

The EAT Manager acts further as a mediate component between the other EAT components and mainly the ACA as well as other packages of the RCL (Figure 5-19).

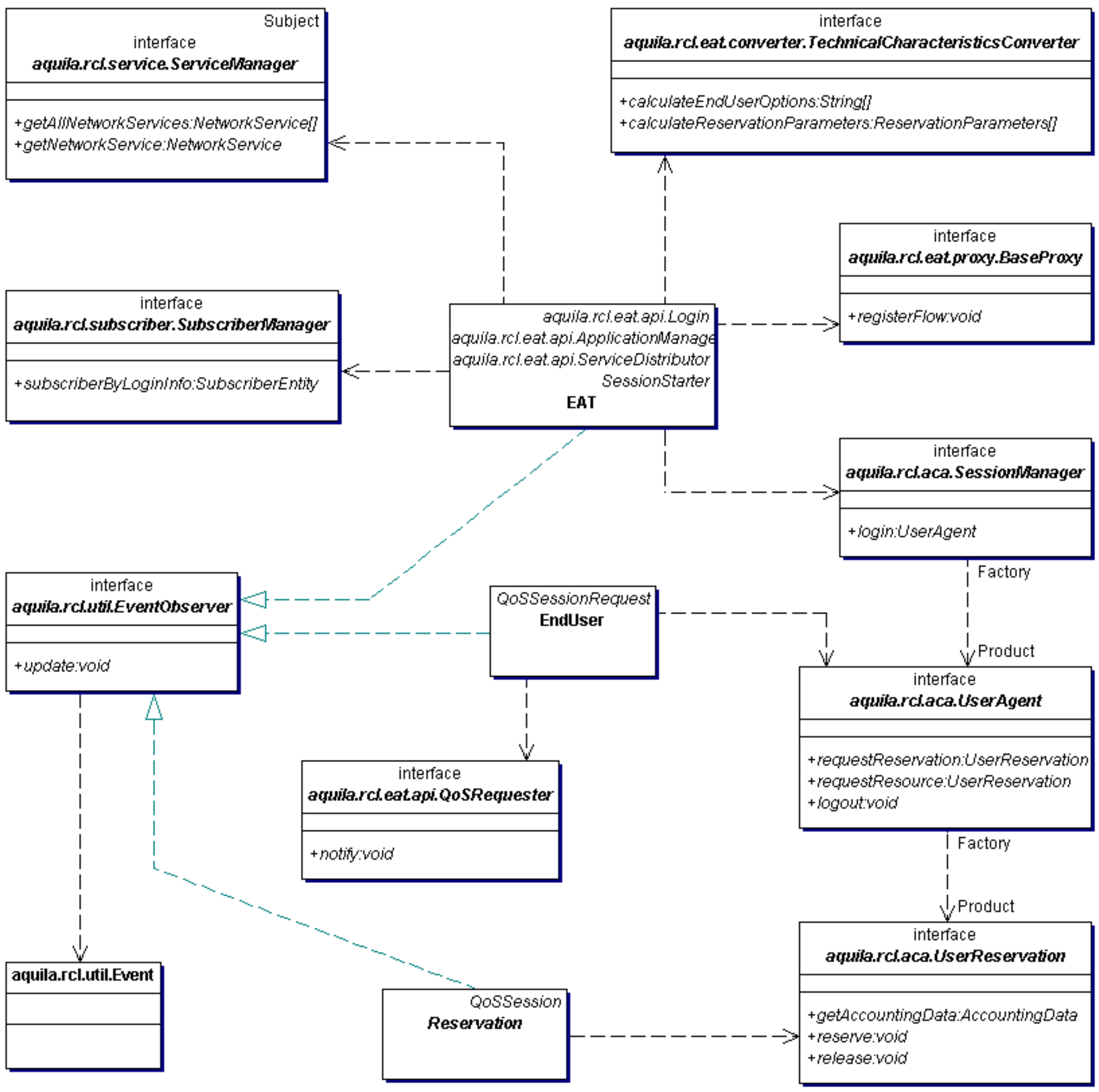
Subject
interface
aquila.rcl.service.ServiceManager
+getAllNetworkServices:NetworkService[]
+getNetworkService:NetworkService
interface
aquila.rcl.eat.converter.TechnicalCharacteristicsConverter
+calculateEndUserOptions:String[]
+calculateReservationParameters:ReservationParameters[]
interface
aquila.rcl.eat.proxy.BaseProxy
+registerFlow:void
interface
aquila.rcl.subscriber.SubscriberManager
+subscriberByLoginInfo:SubscriberEntity
aquila.rcl.eat.api.Login
aquila.rcl.eat.api.ApplicationManage
aquila.rcl.eat.api.ServiceDistributor
SessionStarter
EAT
interface
aquila.rcl.aca.SessionManager
+login:UserAgent
Factory
Product
interface
aquila.rcl.util.EventObserver
+update:void
QoSSessionRequest
EndUser
interface
aquila.rcl.aca.UserAgent
+requestReservation:UserReservation
+requestResource:UserReservation
+logout:void
interface
aquila.rcl.eat.api.QoSRequester
+notify:void
Factory
Product
interface
aquila.rcl.aca.UserReservation
+getAccountingData:AccountingData
+reserve:void
+release:void
aquila.rcl.util.Event
QoSSession
Reservation

Figure 5-19: The by the EAT Manager used external interfaces

5.4.3 GUIs, Converter, and Application Profiles

The legacy application support as well as the support of newly created complex internet services are based on the interworking of the GUIs, the Converter, and the Application Profiles (for more details, see chapter 3.3.1).

In the following diagram, the refined scenario for a usual request is depicted, where all components (also the EAT Manager and the Proxy) are involved:

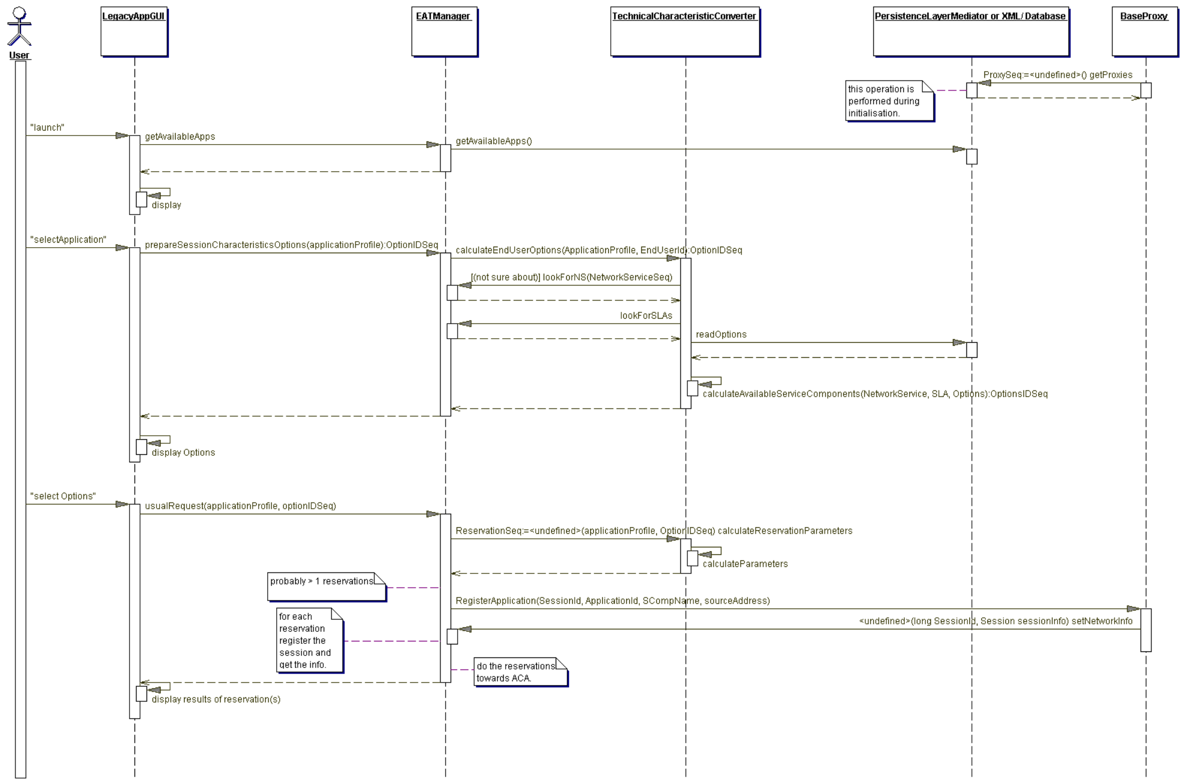
User
LegacyAppGUI
EATManager
TechnicalCharacteristicConverter
PersistenceLayerMediator or XML/ Database
BaseProxy
ProxySeq:=<undefined>() getProxies
this operation is performed during initialisation.
"launch"
getAvailableApps
getAvailableApps()
display
"selectApplication"
prepareSessionCharacteristicsOptions(applicationProfile):OptionIDSeq
calculateEndUserOptions(ApplicationProfile, EndUserId):OptionIDSeq
[(not sure about)] lookForNS(NetworkServiceSeq)
lookForSLAs
readOptions
calculateAvailableServiceComponents(NetworkService, SLA, Options):OptionsIDSeq
display Options
"select Options"
usualRequest(applicationProfile, optionIDSeq)
ReservationSeq:=<undefined>(applicationProfile, OptionIDSeq) calculateReservationParameters
calculateParameters
probably > 1 reservations
RegisterApplication(SessionId, ApplicationId, SCompName, sourceAddress)
<undefined>(long SessionId, Session sessionInfo) setNetworkInfo
for each reservation register the session and get the info.
do the reservations towards ACA.
display results of reservation(s)

Figure 5-20: Sequence diagram for usual request

The Converter will become more important for the 2nd trial. Firstly, it has to look for the existence of appropriate SLAs and network services (as shown above). Secondly and optionally, the Converter will realise the so-called CUSTOM service by mapping requests with given QoS specification to appropriate network services.

5.4.4 Proxy – The Protocol Gateway

The Proxy package mainly consists of a central entity called Proxy Manager and a number of application-level proxies. Those proxies are all sub-classes of an abstract class named Application Proxy. This abstract class provides a standard interface for interaction with the application proxies. The Proxy Manager controls the other classes and communicates with the EAT Manager. The application-level proxies communicate only with the Proxy Manager. The internal architecture of the Proxy as well as its interaction with the EAT Manager module is presented in the following diagram.

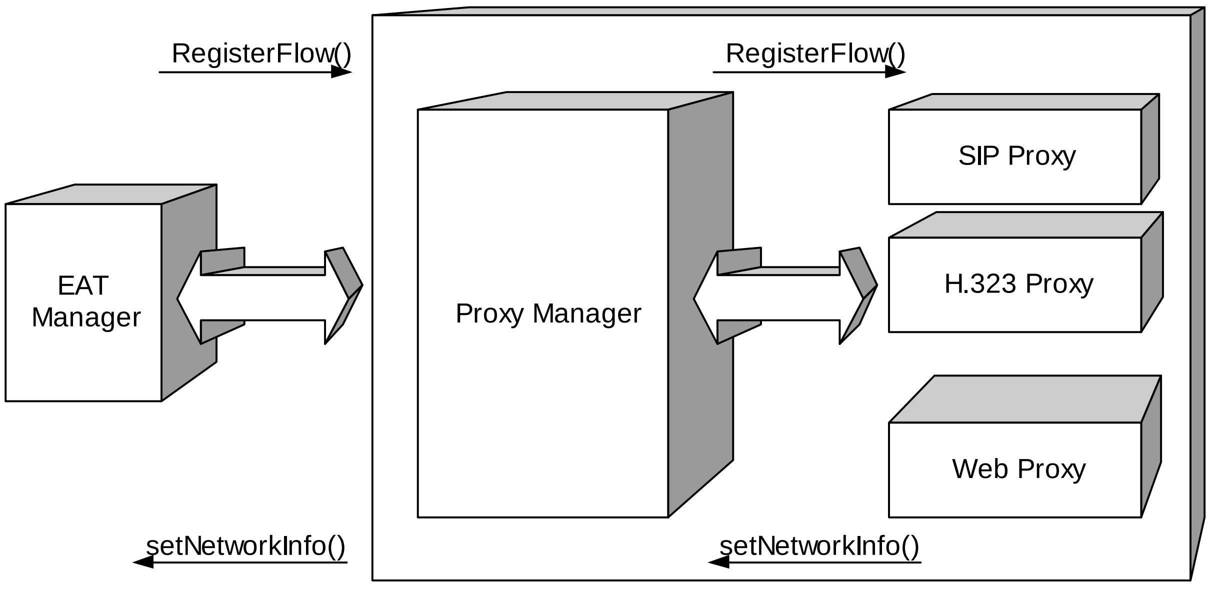

Figure 5-21: Proxy Architecture

5.4.4.1 Interfaces

For the second trial architecture, a small modification of the Proxy interface is foreseen. The Proxy Manager will be invoked by the method:

```
void RegisterFlow(long sessionID, string srcAddr, string destAddr, string serviceComponent, long proxyID)
```

while the Proxy Manager will reply to a request with the method (from eatManager.idl):

```
void setNetworkInfo(long sessionID, Session sessionInfo)
```

where class Session will be enhanced, in order to carry information more than session IP addresses and port numbers.

As already stated, the Proxy may notify the EAT Manager for a flow that has not been already registered. In this case, setNetworkInfo() may be again invoked with a default sessionID.

5.4.5 Deployment Scenario

The EAT will reside in the access network, near to the host, and at the edge of the by the RCL controlled core network (Figure 5-22). It is not necessary to install the EAT on the end-user’s host, because he/she can access the EAT via a web browser. For that, the EAT interacts with a Web server near to it which supports Java Server Pages (JSP) and Java servlets for the dynamically created GUIs/web pages of the EAT.

These GUI servlets access the EAT via the internal (CORBA) API. Also QoS-aware, EAT-based applications may use this API to get QoS from the RCL. (In the case that the general QoS API will be used, this API will be located at the end-user’s host and access the internal EAT API via CORBA, too.)

The Proxy resides also in the access network between the host and the edge router in order to act as a protocol gateway getting the for QoS requests relevant data from application's flow, particularly the control flow (SIP, H.323, RSVP). Between the EAT (Manager) and the Proxy CORBA is used, too, so that they can run on different machines.

The EAT, particularly the Converter, use the application profiles for the mapping of user-friendly QoS terms into RCL-compatible requests. The profiles are accessible via HTTP and can therefore be installed on a central web server or the local one.

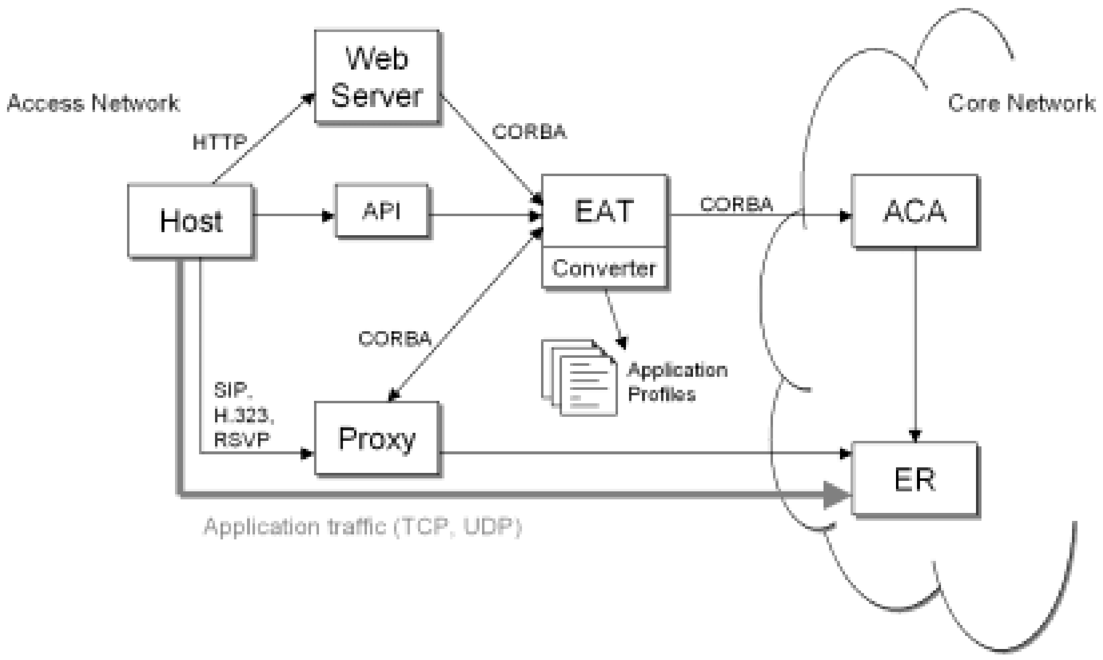

Figure 5-22: Deployment scenario of the EAT components

Generally, there is not need to install an EAT in every access network, since each EAT can act as requester for any host.

For complex internet services (CIS) that provide services with QoS supported by AQUILA, it is reasonable to have an EAT near to this web site (Figure 5-23). Such a CIS, for instance the Mediazine application, acts, on the one hand, as customer/end-user regarding QoS requests towards the RCL, and provides, on the other hand, its media services including QoS features to its human end-users[3].

[3] Concerning the charging & accounting: First of all, the CIS is in charge to pay for the QoS from the RCL. However, it can forward costs to its human customers.

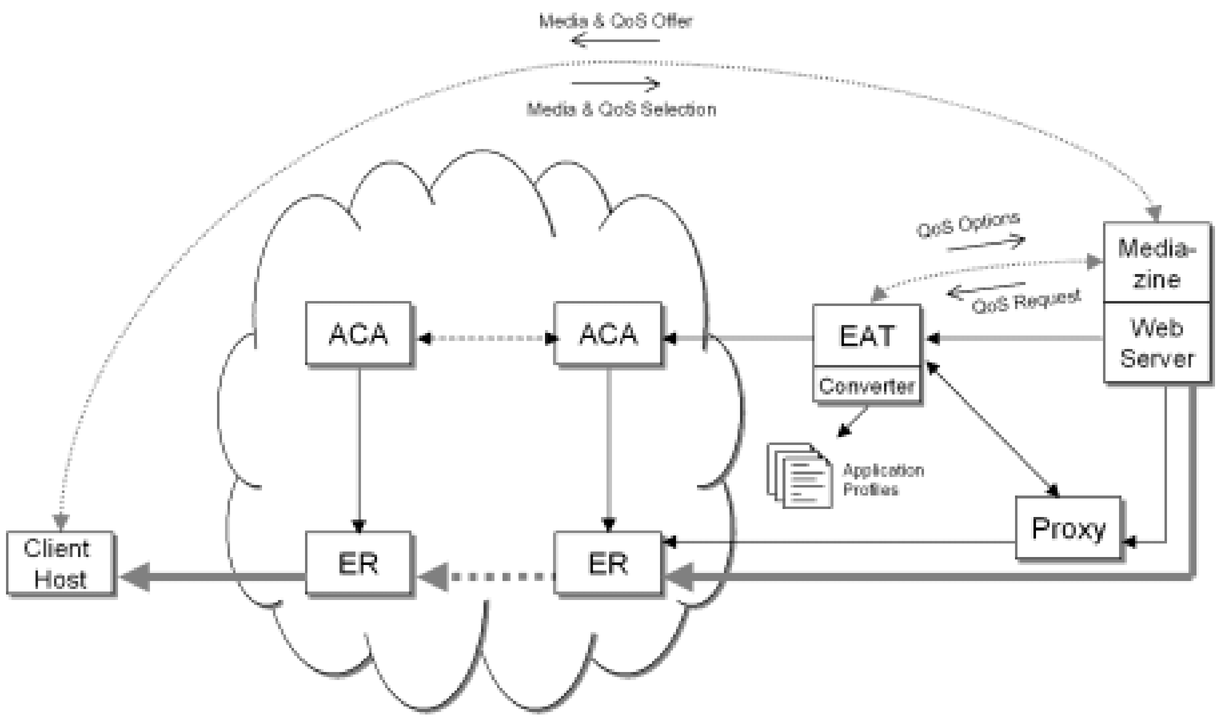

Figure 5-23: Deployment scenario for the MediaZine example

5.5 QMTool

This chapter describes the relations between QMTool and the RCL components for the support of the “monitoring” and the “failure detection” functionality as described in chapter 3.4.

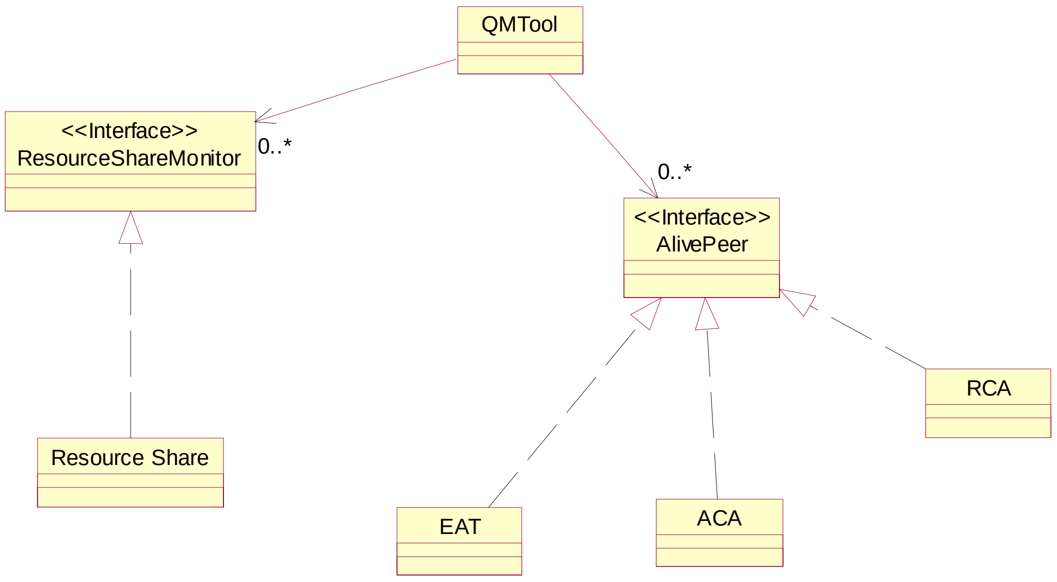

Figure 5-24: QMTool

QMTool, which is the main class, needs to monitor the “reserved” and “total” resources of the Resource Shares belonging to Resource Pools. Therefore, it should maintain references to the monitoring Resource Share objects, which will implement the **ResourceShareMonitor** interface. The latter will enable the retrieval of the monitored resources.

Moreover, it is required that **QMTool** communicates with the **RCA**, **ACA** and **EAT** objects so that it can detect any failure of those components. In the same way as above, it should keep references to these objects, which implement the **AlivePeer** interface.

5.6 Common topics

5.6.1 Prioritised signalling

In the following chapter a coarse design model describes the handling of the prioritised signalling traffic. This basic design model doesn't include attributes and operations, because this is the part of the design specification. Only the basic components with the relations, associations and connections to other basic entities shall be visualised.

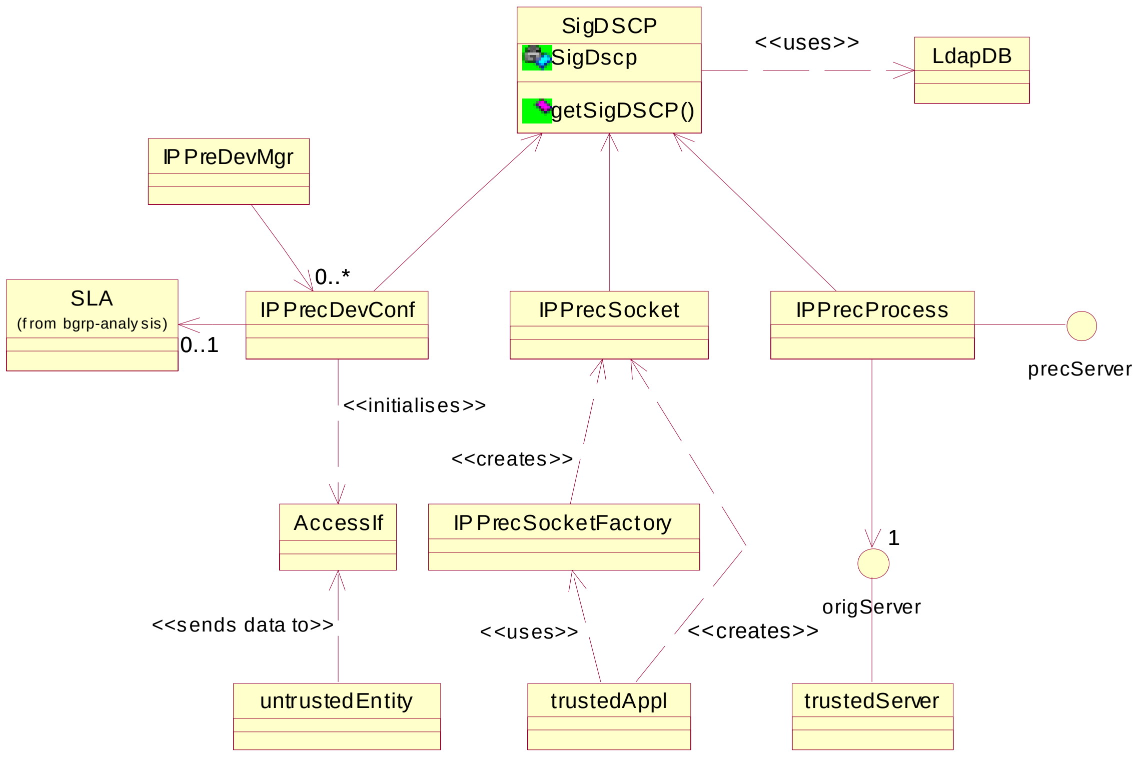

Figure 5-25: Coarse design overview of the three different prioritised signalling cases

The SigDSCP class contains the DSCP type for the signalling traffic available with the getSigDSCP method. The DSCP type itself is stored in the LDAP database.

A precedence mechanism has to be prepared for three different scenarios:

- prioritised traffic of a trusted <u>server</u> located inside the AQUILA domain
- prioritised traffic of a trusted <u>application</u> located inside the AQUILA domain
- prioritised traffic of a untrusted entity <u>outside</u> the AQUILA domain

Therefore three different kinds of implementation exists:

- the **IP Prec**edence **Process**
- the **IP Prec**edence **Socket**
- the **IP Prec**edence **Dev**ice **Conf**iguration

The **IP Precedence Process** uses the **original server interface** (e.g. a server port), offered by a **trusted Server**, in order to catch the server traffic and to set the SigDSCP. The IP Precedence Process itself implements the **IP Precedence Server Interface** (e.g. server port) to send and to receive the prioritised server traffic over this connection point. The IP precedence process also forwards the received traffic at the precedence server interface to the trusted server.

The **IP Precedence Socket** offers a socket mechanism to pre-mark the traffic over this socket. A **trusted application** like the RCL components located inside the AQUILA domain uses this IP Precedence Socket class to set up a pre-marked signalling traffic stream. Two different possibilities of socket instantiation have to be realised. In the direct mode the trusted application itself creates the IP Precedence Socket object. But the trusted application can also use the factory mechanism for a flexible creation of different socket objects. Therefore the **IP Precedence Socket Factory** class provides the instantiation of the IP Precedence Socket object.

An **untrusted Entity**, located outside the AQUILA domain, sends the traffic to an edge router **Access Interface** where the decision is performed to remark the received traffic with a specific DSCP like the SigDSCP or to set the best effort DSCP. Therefore the Access Interface has to be initialised how a specific traffic type like signalling traffic can be identified and remarked. The **IP Precedence Device Configuration** class initialises and configures the Access interfaces of the edge device perhaps in consideration of a Service Level Agreement (SLA) negotiated with the neighbour AS. The task of the IPPrecDevMgr is the co-ordination of the IPPrecDevConf objects.

The class diagram Figure 5-26 shows in a more detailed design model how the IP Precedence Socket class can extend an existing native java net package to support socket-based prioritised signalling and which other classes are also needed.

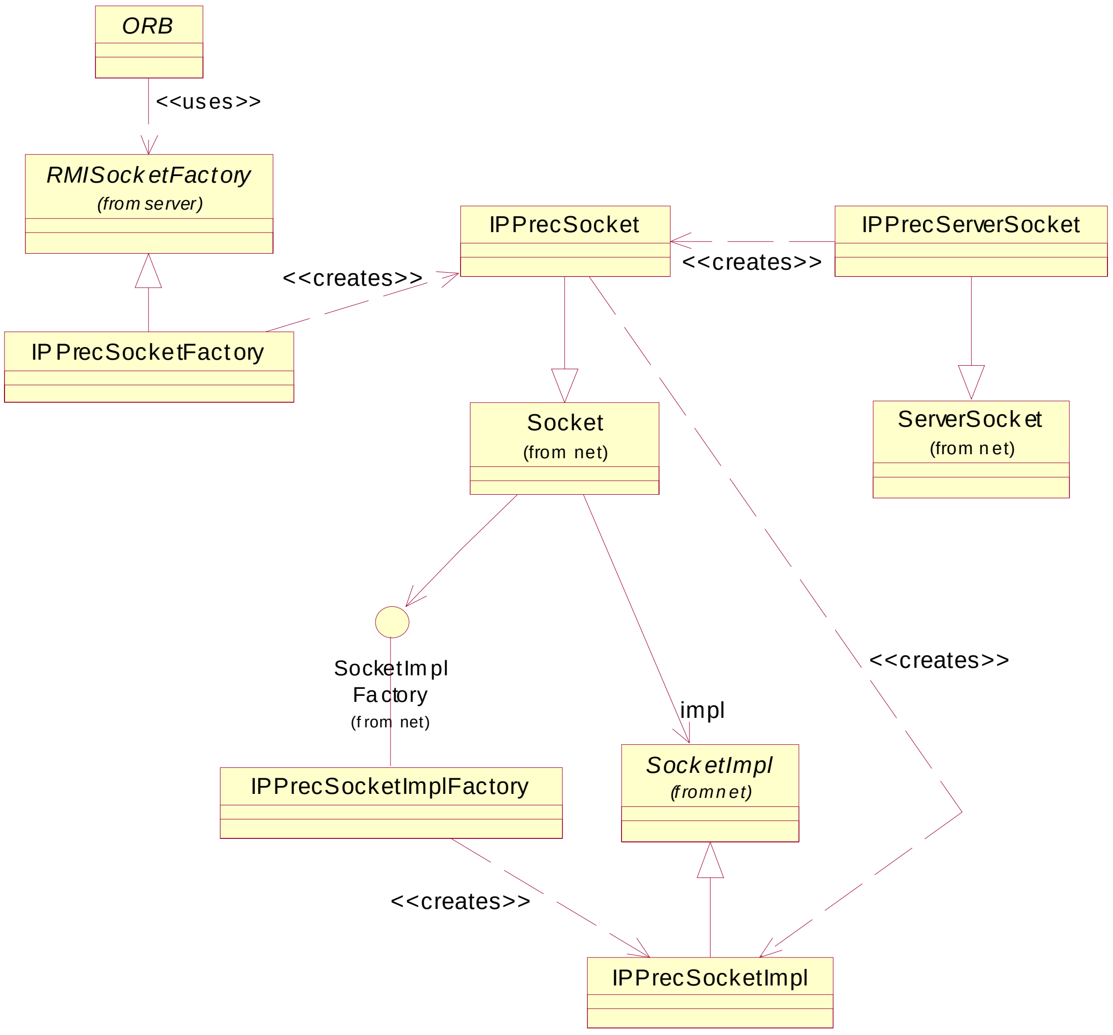

Figure 5-26: Coarse design of the IP precedence signalling handling based on sockets

The **IPPrecSocket** class extends the Socket class of the java net package in order to offer the prioritisation of the traffic over this socket.

The **Socket** class itself uses the abstract **SocketImpl** class with the **PlainSocketImpl** class as default implementation but without any prioritisation. Beside this socket implementation the new **IPPrecSocketImpl** class extends the **SocketImpl** class and offers a new socket implementation extended with the prioritisation issue. The **IPPrecSocket** class itself creates the **IPPrecSocketImpl** class.

So the instantiation of the **IPPrecSocket** object initiates, that traffic through this socket is marked with the **SigDSCP**.

The instantiation of the **IPPrecSocketImpl** class can also be done indirectly with the **SocketImplFactory**. In this case the new **IPPrecSocketImplFactory** class creates the **IPPrecSocketImpl** object. For that purpose this new factory has to be set in the **Socket** class. It should be mentioned that then each new instantiated **Socket** object uses the **IPPrecSocktImpl** class as the socket implementation.

Similar to the socket creation at the client side, the new **IPPrecServerSocket** class extends the **ServerSocket** class and has to create the **IPPrecSocket** class.

The CORBA ORB class uses the factory mechanism in order to create socket objects. Therefore the new **IPPrecSocketFactory** extends the abstract native **RMISocketFactory** class and provides the instantiation of the **IPPrecSocket**. So each CORBA connection can also uses the prioritisation mechanism described above.

Comprising, each trusted entity inside the AQUILA domain, that always has to send precedence traffic, should use the factory mechanism with the new **IPPrecSocketImplFactory** class. Then each CORBA connection of this entity uses automatically this precedence socket mechanism. But if traffic shall be transferred without a prioritisation the other above described mechanism has to be used.

6 Interfaces

6.1 Interfaces of inter-domain resource control

The following figure depicts the message flows for inter-domain reservations. The interfaces to and from the BGRP agents are identified.

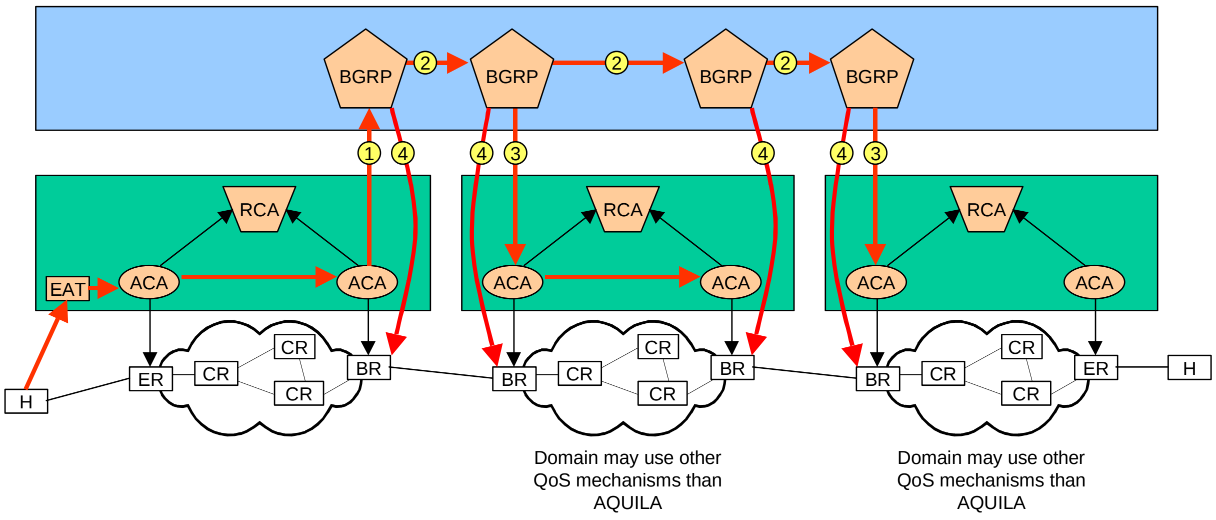
BGRP
BGRP
BGRP
BGRP
2
2
2
1
4
4
3
4
4
3
RCA
RCA
RCA
EAT
ACA
ACA
ACA
ACA
ACA
ACA
H
ER
CR
CR
CR
BR
BR
CR
CR
CR
BR
BR
CR
CR
CR
ER
H
Domain may use other QoS mechanisms than AQUILA
Domain may use other QoS mechanisms than AQUILA

Figure 6-1: Identification of BGRP interfaces

The following interfaces can be identified:

1. Intra-domain resource control sends requests to the BGRP agent
2. BGRP agents communicate with each other (the BGRP protocol)
3. BGRP agents request resources from intra-domain resource control
4. Additionally, BGRP agents have to perform BGP route lookups at their associated border routers.

In the following subchapters, the requirements from each of these interfaces are listed. The interfaces are specified using OMG IDL.

6.1.1 BGRP interface for requests

In the AQUILA architecture, the following entities of the resource control layer are associated with each border router:

- an admission control agent, which manages the resources at the "inner side" of the border router;

- a BGRP agent, which is responsible for inter-domain resources.

These two entities establish a communication association at start-up, which will be used for the interface specified in this subchapter. It is assumed, that both entities trust each other, so that no authentication information will be contained in the messages.

This subchapter specifies the interface, which is offered by BGRP to the ACA, in order to enable the ACA to request inter-domain resources. The following requirements have to be met by this interface:

- BGRP uses globally well known services. So a GWKS indicator will be part of the interface.
- Traffic destination is specified by an IP address prefix.
- To specify the required amount of resources, just a bandwidth parameter is used
- A request will also specify a limit for the perceived QoS in terms of delay, jitter, and packet loss. Additional QoS parameters such as a required ordering may also be specified.

As a result, the requesting ACA should receive the following information:

- The QoS parameters for the path from the egress border router to the ingress border router of the last domain.
- A reference to the intra-domain resource control of the last domain, which may be used to request resources within the last domain from the ingress border router to the egress edge device.
- A reference to a reservation object, which may be used to modify or release the reservation.

These requirements lead to the following IDL specification (the complete IDL file is listed in the Annex):

```
interface BgrpAgent
{
    Reservation request(
        in      Gwks                     service,
        in      IpAddressPrefix          destination,
        in      long long                requiredBW,
        in      service::QoSSpec         requiredQoS,
        out     service::QoSSpec         pathQoS,
        out     DomainResourceManager    destResMgr
    ) raises (BandwidthException, QoSException, RequestException);
};
```

```
interface Reservation
{
    void        modify (
        in      long long               BW
    ) raises (BandwidthException, RequestException);

    void        release ();
};
```

Table 6-1: BGRP interface for requests

6.1.2 Interface between BGRP agents

Each BGRP agent manages a communication association with each neighbour BGRP agents. There are two types of neighbours:

- interior neighbour: a BGRP agent within the same domain;
- exterior neighbour: a BGRP agent within a neighbour domain.

However, the interface between both types is the same. The following requirements take into account the BGRP protocol proposed in [BGRP]. There are five message types: PROBE, GRAFT, ERROR, TEAR, REFRESH.

- All messages are one-way only. Results are not returned for any operation. Instead, another operation in the reverse direction is used. This implies that the return type of all operations is "void" and that all parameters are "in".
- The BGRP agent, which initiates a PROBE message, assigns a unique ID to this message, which enables it to later associate a GRAFT or ERROR message with the PROBE.
- In addition to the minimum required bandwidth, a requested bandwidth will be carried in the PROBE messages. This enables BGRP agents to request additional resources in advance in order to minimise further signalling. The GRAFT message will carry the actually reserved bandwidth.
- As PROBE messages travel along the path, they collect information about the QoS parameters from the initiating BGRP agent to the current location. Also a route record of the path is kept, in order to source route GRAFT or ERROR messages back to the origin. At each stage, the QoS parameters will be updated to include the parameters for the previous hop. The current hop will be added to the route record.
- GRAFT messages will contain a reference to the resource manager of the destination domain, to allow resource reservation for the path from the last domain's ingress border router to the egress edge device.
- To enable quiet grafting, a BGRP agent should be enabled to identify the tree for an incoming PROBE. To support this, the GRAFT message will contain in formation about the

sink tree for a reservation and a list of IP address prefixes, which are covered by this sink tree.

- TEAR and REFRESH messages are exchanged between BGRP agents to lower or confirm resources for a sink tree. The REFRESH also confirms the QoS parameters of the reservation to the root of the sink tree.

These requirements lead to the following IDL specification (the complete IDL file is listed in the Annex):

```
interface BgrpAgent
{
    void        probe(
        in      long                    id,
        in      Gwks                    service,
        in      IpAddressPrefix         destination,
        in      long long               requiredBW,
        in      long long               requestedBW,
        in      service::QoSSpec        requiredQoS,
        in      service::QoSSpec        pathQoS,
        in      RouteRecord             path
    );

    void        graft(
        in      long                    id,
        in      IpAddressPrefix         destination,
        in      Gwks                    service,
        in      long long               reservedBW,
        in      DomainResourceManager   destResMgr,
        in      TreeId                  treeId,
        in      IpAddressPrefixList     addressPrefixList,
        in      service::QoSSpec        pathQoS,
        in      RouteRecord             path
    );

    void        error(
        in      long                    id,
        in      string                  reason,
        in      RouteRecord             path
    );

    void        tear(
        in      TreeId                  treeId,
        in      long long               BW
    );

    void        refresh(
        in      TreeId                  treeId,
        in      long long               BW,
        in      service::QoSSpec        pathQoS
    );
};
```

Table 6-2: Interface between BGRP agents

6.1.3 Intra-domain resource control interface for BGRP requests

At each transit domain, the ingress BGRP agent will contact the intra-domain resource control, to request the resources within the domain from the ingress to the egress border router. In the AQUILA architecture, the ACA associated with the ingress border router will be contacted.

At the destination domain, a similar request will be issued to the intra-domain resource control. However, no egress point will be specified. Instead, the request may be seen as a preliminary action to prepare the intra-domain resource control to accept a request from the initiating domain for the path from the ingress border router to the egress edge device.

The following bullets list the requirements for this interface:

- The interface for these requests should allow a variety of intra-domain resource control mechanisms. Especially, it should not be limited to the AQUILA intra-domain resource control.

- The interface should allow the modification of existing reservations (increase and decrease reserved bandwidth)

```
interface DomainResourceManager
{
    Reservation request (
        in      Gwks                    serviceId,
        in      long long               BW,
        in      IpAddress               entrance,
        in      IpAddress               exit,
        out     service::QoSSpec        domainQoS
    ) raises (BandwidthException, RequestException);
};

interface Reservation; (see Table 6-1 on page 173)
```

Table 6-3: Intra-domain resource control interface for BGRP requests

6.1.4 Interface to BGP

Inter-domain reservations with BGRP follow the BGP routing paths. To do this, BGRP agents require a mechanism to perform BGP route lookups. While a real-world implementation obviously has to use the interfaces provided by the router manufacturers, we describe here an abstract IDL interface for that purpose. An adaptation layer may be used to map that abstract interface to the specific router implementation.

```
interface Bgp
{
    void lookup (
        in      IpAddressPrefix         destination,
        out     IpAddress               nextHop,
        out     boolean                 interiorRoute,
        out     IpAddressPrefixList     nlri
    ) raises (LookupException);
};
```

Table 6-4: Interface for BGR route lookups

6.2 Management interfaces

For the implementation of the "monitoring" and "failure detection" functionalities of the QMTool, the latter should interact with the RCL components that it wants to monitor or detect their status.

The following interfaces can be identified:

1. QMTool sends requests to the RCL elements in order to check whether or not they are alive.

2. QMTool communicates with the resource shares of the Resource Pools and ACAs in order to retrieve their reserved and total resources.

Subsequently, these interfaces are specified with OMG IDL.

6.2.1 Interface between QMTool and RCL components

The RCL components that necessitate their failure detection are the RCA, the ACAs and the EATs. For that reason, they should provide a simple interface to QMTool so that the latter can detect their possible failure. This interface can be the existing one (AlivePeer), which is used for the Alive Peer mechanism among the RCL components. Therefore, the interface offered by the RCL components to QMTool is the following:

```
interface AlivePeer
{
    void hello(in AlivePeer peer, in long seq
    ) raises (AliveException);
};
```

Table 6-5: Interface for failure detection

6.2.2 Interface between QMTool and Resource Shares

In order that QMTool monitors the Reserved and Total resources of the resources pools, each Resource Share of a resource pool should provide an interface to QMTool so that it can retrieve that kind of information. A request to each Resource Share from QMTool should not include any further information. In both cases (reserved and total resources), QMTool will receive a TrafficSpec object that will contain the requested information. The simplification of the TrafficSpec class that is envisaged for the second trial will not affect the impending interface. Its description is shown below.

```
interface ResourceShareMonitor
{
    TrafficSpec getReserved (
    ) raises (MonitoringException);

    TrafficSpec getTotal (
    ) raises (MonitoringException);

};
```

Table 6-6: Interface for Monitoring

6.3 End-user application API

6.3.1 Internal EAT API

The following IDL is a modified version to the existing and implemented one for the first trial. The proposed significant modifications are in detail:

- No longer differentiation between "request levels":
  interfaces `{Advanced, Custom, Usual}QoSSessionRequest` dropped,
  methods moved to interface `QoSSessionRequest`

- New struct `SLA`

- Four new methods in `QoSSessionRequest` to request for session groups:
  `advancedBidirectionlRequest(), advancedGroupRequest(), usualBidirectionalRequest, usualGroupRequest()`

- The interface `QoSSession` is now the "super" interface for the new interfaces `GroupedQoSSession` (including the methods `join()` and `leave()`) and `SingleQoSSession` (that is, the former `QoSSession` interface).

- New struct `MonitoringData`,
  new method `SingleQoSSession.getQoSMonitoringInformation()`

- New interface `QoSRequester` to be implemented by the client in order to be notified when something happens with a former requested reservation.

```
/*
    Package aquila.rcl.eat.api
    File    api.idl
    Version 3.1, 21.09.2001

    Dresden University of Technology (TUD),
    National Technical University of Athens (NTUA)

    This file contains the specification of the (internal) api
    of the End-user Application Toolkit (EAT) for the 2nd trial.

    It is intended for the use by the applications as well as the
    Reservation GUI, etc.
```

```
*/

#ifndef   api_idl
#define   api_idl

#include "service.idl"
#include "subscriber.idl"
#include "tc.idl"
#include "aca.idl"
#include "converter.idl"

module api {

    /* --- Forward declarations --- */

    interface QoSSessionRequest;
    interface QoSSession;
    interface GroupedQoSSession;
    interface QoSRequester;

    typedef sequence<QoSSession> QoSSessionSeq;

    /**
     * APIException containing a message string.
     */
    exception APIException {
        string reason;
    };


    /* --- Login ------------------------------------------------------------ */

    /**
     * The Login interface allows the authentication of an end-user against
     * the EAT (Manager). It is implemented by the EAT.
     * The login information is also forwarded to the ACA.
     */
    interface Login {

        const string SERVER_NAME = "Login";
        const string SERVER_KIND = "EAT login object";

        /**
         * Login at the EAT Manager's API.
         * @param   loginInfo        Account name and password.
         * @param   clientSessionId  Created by a Web server providing the
         *                           Login GUI.
         * @return                   Reference to a new QoSSessionRequest
         *                           object.
         */
        QoSSessionRequest loginClient (
            in subscriber::LoginInfo logInfo,
            in string                clientSessionId)
            raises (APIException);

        /**
         * Gets the reference to the QoSSessionRequest object created at the
         * login.
         * Called by the Reservation GUI, the Usual Reservation GUI, etc.
         * @param   clientSessionId  Web servers' session id, included in the
         *                           URL.
         * @return                   Reference to the existing QoSSessionRequest
         *                           object.
         */
        QoSSessionRequest getQSRequestMeansSessionId (
            in string clientSessionId);
        /**
         * Closes explicitely a client session without logout;
         * destroys the client session id at the EAT Manager.
         * @param   clientSessionId  Web servers' session id, included in the
         *                           URL.
         */
        void close (
            in string clientSessionId)
            raises (APIException);
    };


    /* --- SLA -------------------------------------------------------------- */

    /**
     * The Service Level Agreement between a customer (end-user) and a provider.
     * accountName       The login name of the subscriber.
     * networkServiceId  The id of the subscribed network service.
     * trafficLimit      Optional traffic limitations.
     */
    struct SLA {
        string             accountName;
        service::ServiceID networkServiceId;
        tc::TrafficSpec    trafficLimit; // to be clarified; -1 means "no limit"
                                         // could it be SLS limit, too?
    };
    typedef sequence<SLA> SLASeq;


    /* --- QoS Session Request ---------------------------------------------- */

    typedef sequence<string> ServiceCompSeq;
    typedef sequence<string> ClientSessionIdSeq;

    /**
     * The QoSSessionRequest interface is the user agent for QoS session
     * requests and SLA retrieval.
     * Requests can be made on an advanced and on a usual level.
     */
    interface QoSSessionRequest {

        /**
         * The subscriber.
         */
        readonly attribute string accountName;

        /**
         * All current client session ids.
         */
        readonly attribute ClientSessionIdSeq clientSessionIds;


        /* --- SLAs --- */

        /**
         * The SLAs.
         */
        readonly attribute SLASeq contracts;


        /* --- Advanced Request --- */

        /**
         * For the specification of an advanced request.
         * applicationId     Name that the end-user uses to identify the
         *                   application.
         * sCompName         Service component that this reservation corresponds
     *                     to.
     * networkServiceId  Network service id.
     * reqSLS            Requested SLS incl, scope, flow, traffic spec, QoS
     *                   spec, and schedule.
     * proxyId           Id of the proxy to be used (see ApplicationProxy),
     *                   0 for none.
     */
    struct AdvancedSpec {
        string             applicationId;
        string             sCompName;
        service::ServiceID networkServiceId;
        aca::SLS           reqSLS;           // QoSSpec has only a meaning
                                             // if service id = "CUSTOM"
        long               proxyId;
    };
    typedef sequence<AdvancedSpec> AdvancedSpecSeq;

    /**
     * Requests for a QoSSession on advanced level.
     * It is foreseen for reservation requests that are based
     * on the content of the ACA's reservation request interface.
     * The Reservation GUI uses it, for example.
     * @param  requestSpec  The requested parameters.
     * @param  requester    The requester object that has to be notified
     *                      when something happens with the reservation.
     *                      Can be null.
     * @return              Reference to a new (Single)QoSSession object.
     */
    QoSSession advancedRequest (
        in AdvancedSpec requestSpec,
        in QoSRequester requester)
        raises (APIException);

    /**
     * Advanced request for a bi-directional reservation,
     * building a group of QoS sessions with permutted
     * sender and destination addresses.
     * Complete group is automatically requested!
     * @param  requestSpec  The requested parameters.
     * @param  requester    The requester object that has to be notified
     *                      when something happens with the reservation.
     *                      Can be null.
     * @return              Reference to a new (Grouped)QoSSession object.
     */
    QoSSession advancedBidirectionalRequest (
        in AdvancedSpec requestSpec,
        in QoSRequester requester)
        raises (APIException);

    /**
     * Advanced request for a group of QoS sessions,
     * @param  requestSpecs           An array of request specifications.
     * @param  complete_group_needed  If yes, all or no reservations are
     *                                accepted.
     * @param  requester              The requester object that has to be
     *                                notified when something happens with
     *                                the reservation. Can be null.
     * @return                        An new (Grouped)QoSSession object,
     *                                containing one SingleQoSSession
     *                                object for each single request.
     */
    QoSSession advancedGroupRequest (
        in AdvancedSpecSeq requestSpecs,
        in boolean         complete_group_needed,
        in QoSRequester    requester)
        raises (APIException);
```

```
/* --- Usual Request --- */

/**
 * For the specification of a usual request.
 * applicationProfile  URI of the associated application profile.
 * applicationId       The application name within the application
 *                     profile.
 * selection           Selected session characteristic options from the
 *                     profile.
 * sCompName           The service components that the selection
 *                     corresponds to (same order as in selection).
 * reqScope            Scope: reservation style.
 * reqFlow             Source and dest. addresses, ports, protocol,
 *                     DSCP.
 * schedule            Service schedule: reservation time.
 * proxyName           Name of the proxy to be used (see application
 *                     profile), the EAT Manager can look into the *.rc
 *                     file in order to get the id of the proxy;
 *                     "" if no proxy is needed.
 */
struct UsualSpec {
    converter::URI         applicationProfile;
    string                 applicationId;
    converter::OptionIDSeq selection;
    ServiceCompSeq         sCompName;
    aca::Scope             reqScope;
    aca::Flow              reqFlow;
    aca::ServiceSchedule   schedule;
    string                 proxyName;
};
typedef sequence<UsualSpec> UsualSpecSeq;

/**
 * Prepares the Usual App GUI with the options from the profile,
 * gets the ids for the session characteristics options to be displayed.
 * @param  applicationProfile  URI of the associated application
 *                             profile.
 * @return                     An array of option ids of the profiles.
 */
converter::OptionIDSeq prepareSessionCharacteristicsOptions (
    in converter::URI applicationProfile)
    raises (APIException);

/**
 * Requests for a QoSSession on usual level.
 * It is foreseen for applications which are not QoS-aware
 * but are supported by application profiles for manual,
 * non-professional reservations. The end-user has to ask for the
 * preparation of suitable Session Charactersitics options and requests
 * then for such as QoS session.
 * @param  requestSpec  The requested parameters.
 * @param  requester    The requester object that has to be notified
 *                      when something happens with the reservation.
 *                      Can be null.
 * @return              Reference to a new QoSSession object, may be
 *                      group object, containing one session for each
 *                      service component, or a single session object.
 */
QoSSession usualRequest (
    in UsualSpec    requestSpec,
    in QoSRequester requester)
    raises (APIException);

/**
 * Usual request for a bi-directional reservation,
 * building a group of QoS sessions with permutted
```

```
         * sender and destination addresses.
         * Complete group is automatically requested!
         * @param  requestSpec  The requested parameters.
         * @param  requester    The requester object that has to be notified
         *                      when something happens with the reservation.
         *                      Can be null.
         * @return              Reference to a new (Grouped)QoSSession object,
         *                      containing either two (sub) groups one for each
         *                      participant, or containing two single sessions
         *                      one for each participant.
         */
        QoSSession usualBidirectionalRequest (
            in UsualSpec    requestSpec,
            in QoSRequester requester)
            raises (APIException);

        /**
         * Usual request for a group of QoS sessions,
         * for example for bi-directional reservations.
         * @param  requestSpecs           An array of request specifications.
         * @param  complete_group_needed  If yes, all or no reservations are
         *                                accepted.
         * @param  requester              The requester object that has to be
         *                                notified when something happens with
         *                                the reservation. Can be null.
         * @return                        Reference to a new (Grouped)QoSSession
         *                                object, containing either other (sub)
         *                                groups for the participants, or
         *                                containing single session objects
         *                                one for the participants.
         */
        QoSSession usualGroupRequest (
            in UsualSpecSeq requestSpecs,
            in boolean      complete_group_needed,
            in QoSRequester requester)
            raises (APIException);


        /* --- Common --- */

        /**
         * Returns all active reservations.
         * @return  Sequence of QoSSession objects.
         */
        QoSSessionSeq getActiveQoSSessions ();

        /**
         * Logs the end-user out, releases all reservations.
         */
        void logout ()
            raises (APIException);
    };


    /* --- QoS Session ------------------------------------------------------ */

    /**
     * SessionStatus indicates whether a requested (and by the EAT accepted)
     * reservation is still provisional (waiting for Proxy response) or already
     * accepted by the ACA and therefore active.
     * (Rejected reservations are immediately released, and the client is
     * informed.)
     */
    enum SessionStatus {
        Provisional,
        Active
    };
```

```
/**
 * MonitoringData belongs to the retrieval of QoS monitoring information.
 * The content is still undefined.
 */
struct MonitoringData {
    long dummy;
};

/**
 * A QoSSession can be single or a grouped session.
 */
interface QoSSession {

    /**
     * Is this a grouped or a single session?
     */
    readonly attribute boolean isGroup;


    /**
     * The aggregating group. Can be null.
     */
    readonly attribute GroupedQoSSession group;


    /**
     * The requester of this session, can be null.
     */
    readonly attribute QoSRequester requester;

    /**
     * Releases the associated single reservation or the whole group.
     */
    void release ()
        raises (APIException);
};

/**
 * The GroupedQoSSession interface belongs to a reservation group,
 * e.g. a bi-directional reservation.
 */
interface GroupedQoSSession : QoSSession {

    /**
     * The QoS sessions. Can be either other groups or single sessions.
     */
    readonly attribute QoSSessionSeq sessions;

    /**
     * If yes, the group consists of two sessions with the same SLS but
     * permuted sender and destination addresses.
     */
    readonly attribute boolean isBidirectional;

    /**
     * Returns one specific QoSSession object of the group.
     * @param  sessionId  The session id of the reservation.
     * @return            The QoSSession object with the specified id.
     */
    QoSSession getSession (in long sessionId);

    /**
     * Adds/joins an already established, single QoSSession to this group.
     * @param  sessionId  The id of the session to be added.
     */
    void join (in long sessionId)
```

```
            raises (APIException);

        /**
         * Removes a QoSSession object from this group without releasing it.
         * @param  sessionId  The if of the session to be removed.
         */
        void leave (in long sessionId)
            raises (APIException);
    };

    /**
     * The SingleQoSSession interface belongs to an actual reservation.
     */
    interface SingleQoSSession : QoSSession {

        readonly attribute long                sessionId;
        readonly attribute SessionStatus       status;

        // Content of the advanced/usual request:
        readonly attribute converter::URI      applicationProfile; // "" if none
        readonly attribute string              applicationId;
        readonly attribute converter::OptionID selection;          // "" if none
        readonly attribute string              sComponentName;
        readonly attribute service::ServiceID  networkServiceId;
        readonly attribute aca::SLS            reqSLS;
        readonly attribute long                proxyId;

        /**
         * Retrieves the accounting data of this session.
         * Automatically done before release.
         * @return  Accounting information of the current session.
         */
        service::AccountingData getAccountingInformation ();

        /**
         * Retrieves the QoS monitoring data of this session.
         * @return  Monitoring information of the current session.
         */
        MonitoringData getQoSMonitoringInformation ();
    };


    /* --- QoS Requester ---------------------------------------------------- */

    /**
     * RequestEvent describes what with a reservation (request) ca be happen:
     * A provisional reservation can be accepted or rejected.
     * An already established reservation can broke, e.g. when it brokes or
     * when the schedule finishes.
     */
    enum RequestEvent {
        Accepted,
        Rejected,
        Released
    };

    /**
     * The QoSRequester interface should be implemented by the QoS requesting
     * client in or der to have the chance to inform it about some events
     * concerning the requested reservation, e.g. when a provisional reservation
     * has been accepted/rejected by the ACA, or when a reservation brokes, etc.
     */
    interface QoSRequester {

        /**
         * Notifies the requesting client about something that happens with the
         * reservation.
         * @param  sessionId  The id relevant QoSSession.
         * @param  ev         The occuring event.
         * @param  reason     The optional reason for the event.
         */
        void notify (
            in long         sessionId,
            in RequestEvent ev,
            in string       reason);
    };


    /* --- Application Manager --------------------------------------------- */

    /**
     * ApplicationInformation contains the URI of the associated app profile,
     * the app's name, the version, and the build no.
     */
    struct ApplicationInformation {
        converter::URI   aplicationProfile;
        string           applicationName;
        string           versionNo;
        string           buildNo;
    };

    typedef sequence<ApplicationInformation> ApplicationInformationSeq;

    /**
     * ApplicationProxy contains all proxy information, such as the id,
     * the name, the description, and may be the control port, just to show it.
     */
    struct ApplicationProxy {
        long   proxyId;
        string proxyName;
        string proxyDescription;
        long   controlPort;
    };

    typedef sequence <ApplicationProxy> ApplicationProxySeq;

    /**
     * The ApplicationManager interface provides information about installed
     * available application (profiles), application proxies, etc.
     */
    interface ApplicationManager {

        const string SERVER_NAME = "ApplicationManager";
        const string SERVER_KIND = "EAT application manager object";

        /**
         * Gets all available apps for the Legacy App GUI.
         * @return  Sequence of ApplicationInformation objects.
         */
        ApplicationInformationSeq getAvailableApps ();

        /**
         * Gets all installed proxies for the Reservation GUI.
         * @return  Sequence of ApplicationProxy objects.
         */
        ApplicationProxySeq getApplicationProxies ();
    };


    /* --- Network Service Distributor -------------------------------------- */

    /**
     * The ServiceDistributor interface provides information about available
     * network services, and which ones are included in the current SLA.
     */
    interface ServiceDistributor {

        const string SERVER_NAME = "ServiceDistributor";
        const string SERVER_KIND = "EAT service distributor object";

        /**
         * Gets all available network services.
         * @return  Sequence of NetworkService objects.
         */
        service::NetworkServiceSeq getNetworkServices ();
    };
};

#endif // api_idl
```

Part IV: Abbreviations and references

7 Abbreviations

A

ACA	Admission Control Agent
API	Application Programming Interface
AS	Autonomous System

B

BGP	Border Gateway Protocol
BGRP	Border Gateway Reservation Protocol
BR	Border Router

C

CBR	Constant Bit Rate
CDN	Content Delivery Network, Content Distribution Network
CIDR	Classless Inter Domain Routing
CLI	Command Line Interface
CORBA	Common Object Request Broker Architecture
CPE	Customer Premises Equipment
CR	Core Router

D

DB	Database
DiffServ, DS	Differentiated Services
DSCP	DiffServ Code Point

E

EAT	End-user Application Toolkit
ED	Edge Device
ER	Edge Router

I

IETF	Internet Engineering Task Force

IPDV	Instantaneous Packet Delay Variation
IPPM	IP Performance Metrics
ISP	Internet Service Provider

L

LER	Label Edge Router
LSP	Label Switched Path
LSR	Label Switching Router

M

MPLS	Multi Protocol Label Switching

O

OWD	One Way Delay

P

PL	Packet Loss
PCBR	Premium Constant Bit Rate
PMC	Premium Mission Critical
PMM	Premium Multimedia
PVBR	Premium Variable Bit Rate

Q

QoS	Quality of Service

R

RCA	Resource Control Agent
RCL	Resource Control Layer
RP	Resource Pool
RPL	Resource Pool Leaf
RPT	Resource Pool Tree
RTD	Round Trip Delay

S

SIBBS	Simple Inter Bandwidth Broker Signalling
SIP	Session Initiation Protocol
SLA	Service Level Agreements
STD	Standard

T

TCL	Traffic Class
TCP	Transmission Control Protocol

V

VBR	Variable Bit Rate
VPN	Virtual Private Network

W

WFQ	Weighted Fair Queuing

8 References

[12sag1014a] T. Engel: “Control-loops”, IST-1999-10077-WP1.3-SAG-01014-TC-CC/a

[13cor1015a] F. Ricciato, S.Salsano, G. Troiano: Control Loops for Provisioning, IST-1999-10077-WP1.3-COR-01015-TC-CC/a

[13cor1016b] S. Mastropietro, F. Ricciato, S. Salsano, G. Troiano: Evaluation of distributed Admission Control schemes, IST-1999-10077-WP1.3-COR-01016-TC-CC/b

[13ntu1025a] Two aspects for the second trial: Simplification of Traffic Spec & Shifting Resources between TCLs, IST-1999-10077-WP1.3-NTU-01025-TC-CC/a

[13sag01010a]Thomas Engel, “A Performance Analysis of Resource Pool”.

[13sag1027a] T. Engel: “Resource Pools for MBAC”

[ABOBA] Aboba, B. and D. Lidyard, "The Accounting Data Interchange Format (ADIF)", Work in Progress.

[ALTM] J. Altmann, and P. Varaiya, "INDEX Project: User Support for Buying QoS with regard to User's Preferences," IWQOS'98, Sixth IEEE/IFIP International Workshop on Quality of Service, pp. 101-104, Napa, USA, May 1998. http://www-networking.eecs.berkeley.edu/~altmann/frames/publications_frame.html

[BGRP] BGRP: Sink-Tree-Based Aggregation for Inter-Domain Reservations Ping P. Pan, Ellen L. Hahne, and Henning G. Schulzrinne, KICS 2000

[BGRPFRM] Pan, E. Hahne, H. Schulzrinne, “BGRP: A Framework for Scalable Resource Reservation”, Internet Draft, <draft-pan-bgrp-framework-00.txt>, January 2000.

[BISSEL] Torsten Bissel, Manfred Bogen, Christian Bonkowski, and Dieter Strecker. QoS Assessment and Measurement for End-to-End Services. Proceedings of the First COST 263 In-ternational Workshop on Quality of Future Internet Services (QofIS 2000). Jon Crowcroft, James Roberts, Mikhail I. Smirnov (Eds.), Lecture Notes in Computer Science 1922, Springer-Verlag, ISBN 3-540-41076-7, pg. 194-207, 2000.

[BLAN] J. Blanquer, J. Bruno, E. Gabber, M. Mcshea, B. Özden, and A. Silberschatz, "Resource Management for QoS in Eclipse/BSD", Proceedings of the FreeBSD 1999 Conference, Berkeley, California, October 1999. http://www.bell-labs.com/project/eclipse/pub.html

[BOGEN] Manfred A. Bogen. A Framework for Quality of Service Evaluation in Distribuded Envi-ronments. Ph.D. Dissertation, University of Nottingham, UK, September 2000, GMD Re-search Series, ISBN 3-88457-384-5, 2001.

[BROW] Brownlee, N. and Blount, A., “Accounting Attributes and Record Formats”, Network Working Group, RFC 2924, September 2000

[BRUS] Jos'e Brustoloni, Eran Gabber, Abraham Silberschatz and Amit Singh, “Quality of Service Support for Legacy Applications”, in Proceedings of the 9th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'99), Basking Ridge, NJ, June 1999, pp. 3-11. http://www.bell-labs.com/user/jcb/ and http://www.bell-labs.com/project/eclipse/

[CMS] P. Calhoun, W. Bulley, S. Farrell, “Diameter CMS Security application”, draft-ietf-aaa-diameter-cms-sec-02.txt (work in progress), July 2001.

[COMA] Esteve Majoral-Coma, Xavier Martínez-Álvarez, Angel Luna-Lambies, Jordi Domingo-Pascual, “Design, implementation and test of a RSVP Agent based on a generic QoS API”, Universitat Politècnica de Catalunya, Advanced Broadband Communication Center, http://www.fokus.gmd.de/events/qofis2000/html/abstracts.html#COM0900:Design http://www.fokus.gmd.de/events/qofis2000/html/programme.html

[CORBASEC]“Security Service Specification” V 1.7 Object Management Group, March 2001

[D1201] IST-1999-10077-WP1.2-SAG-1201-PU-O/b0, System architecture and specification for the first trial

[D1301] IST-1999-10077-WP1.3-COR-1301-PU-O/b0, Specification of traffic handling for the first trial

[D1302] IST-1999-10077-WP1.3-COR-1302-RE-O/*, Specification of traffic handling for the second trial

[D2201] IST-1999-10077-WP2.2-TUD-2201-RE-O/b0, Specification of End-user Application Toolkit

[D3201] IST-1999-10077-WP3.2-TPS-3201-PU-R/b0, First Trial Report

[DeployMCast] “Deployment Issues for the IP Multicast Service and Architecture”, IEEE Network – January/February 2000

[Dov01] C. Dovrolis et al, “What do Packet Dispersion Techniques Measure?”, Infocom 2001, http://www.caida.org/outreach/papers/consti.pdf.

[DSSEC] Z. Fu, S. F. Wu, T.S. Wu, H. Huang, F. Gong, “Security issues for Differentiated Service Framework”, Internet draft, <draft-fu-diffserv-security-00.txt>, October 1999.

[ETSI321] ETSI TS 101 321 V2.1.1, "Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); Inter-domain pricing, authorization, and usage exchange", August 2000

[ETSI734] ETSI TR 101 734- v1.1.1, “Internet Protocol (IP)based networks: Parameters and mecha-nisms for charging”, September 1999

[FANKH] Fankhauser, G., Stiller, B. and Plattner, B., “Arrow: A Flexible Architecture for an Ac-counting and Charging Infrastructure in the Next Generation Internet”, 1st Berlin Internet Economics Workshop, Berlin, October, 1997

[I2-QoS-API] B. Riddle, A. Adamson: A Quality of Service API Proposal, Internet2: Joint Applications/Engineering QoS Workshop, Santa Clara, USA, May 1998, http://www.Internet2.edu/qos/may98Workshop/html/apiprop.html

[ICC01] E. Nikolouzou, G. Politis, P. Sampatakos, “An Adaptive Algorithm for Resource Management in a Differentiated Services Network”

[IETF] http://www.ietf.org

[ISABEL] ISABEL http://isabel.dit.upm.es/

[JOoSAPI] Java QoS API, http://www-vs.informatik.uni-ulm.de/soft/JavaQoS/

[Kassler99] A. Kassler, H. Christein, P. Schulthess: A Generic API for Quality of Service Networking based on Java, in: Proceedings of the ICC'99, Vancouver, Canada, June 1999

[MBAC] W. Burakowski, A. Bak, “Measurement-Based Admission Control (MBAC)”, IST-1999-10077-WP1.3-WUT-01007-TC-CC/a.

[MILLS] Mills, C., Hirsch, G. and G. Ruth, "Internet Accounting Background", RFC 1272, Novem-ber 1991.

[MINDCTI] http://www.mindcti.com/fs/ipservbil_fs.html

[MSDP] Multicast Source Discovery Protocol (MSDP), draft-ietf-msdp-spec-06.txt

[OSS-QoS] JSR #000090, OSS Quality of Service API, http://java.sun.com/aboutJava/communityprocess/jsr/jsr_090_qos.html

[QoSME] P. Wang: QoSME: Quality of Service Management Environment, http://www.cs.columbia.edu/dcc/qosockets/

[QOSWG] Quality of Service Internet2 Working Group http://www.internet2.edu/qos/wg/

[QuAL] P. G. S. Florissi: QuAL: The Quality of Service Assurance Language, http://www.cs.columbia.edu/~pgsf/qual.html

[RFC1519] IETF RFC 1519:Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy

[RFC2058] C. Rigney, A. Rubens, W. Simpson, S. Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2058, January 1997.

[RFC2059] C. Rigney, "RADIUS Accounting", RFC 2059, January 1996.

[RFC2283] Multiprotocol Extensions for BGP-4

[RFC2330] V. Paxon, G. Almes, J. Mahdavi, M. Mathis: Framework for IP Performance Metrics. RFC 2330. February 1998. http://www.ietf.org/rfc/rfc2330.txt

[RFC2362] Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification

[RFC2402] S. Kent, R. Atkinson, "IP Authentication Header", RFC2402, November 1998.

[RFC2406] S. Kent, R. Atkinson, " IP Encapsulating Security Payload (ESP)", RFC2406, November 1998.

[RFC2474] Nicols, K., Blake, S., Baker, F. and D. Black, "Definition of the Differentiated Service Field (DS Field) in the IP v4 and Ipv6 Headers", RFC 2474, December 1998.

[RFC2543] M. Hadley et al., RFC 2543 – Session Initiation Protocol

[RFC2638] K. Nichols, V. Jacobson, L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", July 1999

[RFC2748] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry, "The COPS (Common Open Policy Service) Protocol", RFC 2748, January 2000.

[RFC2829] M. Wahl, H.Alvestrand, J. Hodges, R. Morgan, "Authentication Methods for LDAP", RFC 2829, May 2000.

[RFC2924] Brownlee, N., Blount, A., "Accounting Attributes and Record Formats", RFC 2924, September 2000.

[SABA2] SABA2 http://www.ccaba.upc.es/projects/saba2/E_saba2local.html

[SIBBS] QBone Bandwidth Broker Architecture, Work in Progress, http://sss.advanced.org/bb/bboutline2.html

[SIPRES] W. Marshall et al., Internet draft: draft-ietf-sip-manyfolks-resource-01

[STILLER] Stiller, B., “Overview of Billing Systems for Internet Service Providers”: Billing Software and System Solutions