A QoS Middleware between Users, Applications and the Network

Charilaos Tsetsekas, Sotirios Maniatis
National Technical University of Athens
Department of Electrical and Computer Engineering
Telecommunications Laboratory
9 Heroon Polytechniou str, 15773 Athens, Greece
tel.: + 30 1 772 2424 / fax + 30 1 772 2534
e-mail: htset@telecom.ntua.gr , sotos@telecom.ntua.gr

Falk Fünfstück, Anne Thomas
Technische Universität Dresden
Fakultät Informatik
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
D-01062 Dresden, Germany
tel. + 49 351 463 8555 / fax + 49 351 463 8459
e-mail: Falk.Fuenfstueck@inf-tu-dresden.de , Anne.Thomas@inf-tu-dresden.de

Yannis Karadimas
Q-Systems S.A.
84 Vas. Sofias Ave., 11528 Athens, Greece
tel.: +30 1 748 9891 / fax + 30 1 779 5194
e-mail: jkar@qsystems.gr

Abstract: *The support of Quality of Service in the Internet has become one the most important topics within the Internet community. The introduction of the Integrated Services and the Differentiated Services architectures was a major breakthrough in this direction. Enhanced by the Bandwidth Broker concept, DiffServ aims to provide QoS in the Internet through the prioritisation of some IP flows over others. One of the major shortcomings of the DiffServ architecture is the lack of a standard mechanism for the interaction between users/applications and the Bandwidth Brokers, so that end-to-end QoS is achieved.*
In this paper we present a distributed architecture for the provision of QoS in the Internet, as specified within the IST Aquila project. However, the paper focuses on the bridging of the gap that currently exists between applications and the network and presents the End-user Application Toolkit (EAT). The EAT middleware provides a framework for the presentation of network services to users, the description and selection of QoS parameters and the forwarding of reservation requests. Through the concept of Application Profiles, it aims to support QoS for legacy applications, that is, commercial applications that cannot be modified to support QoS.

Keywords: Quality of Service, Differentiated Services, Resource Control Layer, End-user Application Toolkit, Application Profiles, Protocol Gateways, QoS API

1 Introduction

The Internet continues to evolve into a network that supports a wide spectrum of applications with diverse requirements. The introduction of voice and multimedia in general poses delay, throughput and packet loss requirements towards the network and makes the provision of Quality of Service a very important issue. The Integrated Services architecture [IntServ] was the first significant step for the introduction of QoS in the Internet. In IntServ, the exchange of Resource Reservation Protocol (RSVP) [RSVP] messages reserves network resources on each node along the path from the sender to the receiver. IntServ is able to provide QoS guarantees,

but the fact that a separate reservation should be made for each requesting flow raised fears for scalability inefficiency. Those concerns were also fuelled by the constant RSVP message exchange as well as the need for keeping reservation state on each router.
The Differentiated Services architecture [DiffServ] emerged as the solution to the scalability problem of IntServ. In DiffServ, instead of establishing separate reservations for each IP flow, flows with similar traffic characteristics and QoS needs are grouped together under a common IP header field and handled in the same queue in the routers in the core network. In this way, the role of the core routers is reduced to classifying packets according to this field, the DiffServ Code Point (DSCP), into the appropriate aggregate queue. The Differentiated Services architecture is enhanced with the introduction of the concept of the Bandwidth Broker [BB], a network entity that centrally manages the resources of an administrative domain and allocates network resources to requesting hosts.
The Differentiated Services architecture coupled with the Bandwidth Broker entity presents a more efficient and manageable solution for the provision of QoS in the Internet than IntServ. However, it faces some problems and limitations. A centralized Bandwidth Broker for an administrative domain may become a bottleneck and a single point of failure in the QoS network. A more distributed architecture is needed to this direction. Moreover, currently a gap exists between users and applications, in the sense that no unified mechanisms exist for the users to make reservation requests to the QoS network. Where such mechanisms exist, for instance RSVP in IntServ, they are difficult to use by normal users.
In this paper, we briefly present a distributed architecture for the provision of QoS in a DiffServ-based IP network. The main innovation is the introduction of a new layer on top of DiffServ, the Resource Control Layer (RCL) [RCL]. The RCL, in a sense, resembles a distributed Bandwidth Broker in managing the resources of the core network and distributing resource shares to requesting users and applications. However, the paper focuses on a vital part of the RCL, the End-user Application Toolkit.
The End-user Application Toolkit (EAT) is a web-based architecture residing at the edge of the network. It aims to cover the gap between users/applications and the network as far as sending reservation requests is concerned. Nevertheless, the EAT does more than just perform QoS signalling operations. The main objective of the EAT is the transparent QoS support for legacy applications. With the term “legacy applications” we refer to popular non-QoS aware applications that cannot be modified (e.g. no source code is available) to use a QoS API. For those applications, Application Profiles are created that capture their individual QoS requirements, in a way understandable to the end-user. The EAT provides web-based graphical user interfaces (GUIs) that the users can access with their web browsers. Those GUIs are dynamically created for each selected application, based on its Application Profile, and present to users all the available options with regard to QoS, using higher level semantics. Finally, the EAT provides an abstraction from different reservation mechanisms; although a proprietary protocol based on CORBA [CORBA] is used for the origination of request to the RCL, the EAT transparently supports also applications that already support RSVP as a QoS signalling protocol.
In the next paragraph we present in brief the AQUILA architecture, an architecture for dynamic end-to-end QoS provision in the Internet, as it is being defined, implemented and tested in the framework of the IST AQUILA project [AQUILA]. The main concepts of the architecture are presented and its components are described. Then, we focus on the End-user Application Toolkit by presenting the problems that it aims to solve and the components responsible for this.

2 The AQUILA network

The aim of this section is to give a short introduction of the AQUILA architecture. The AQUILA architecture consists of two functional areas: the data plane that is responsible for transmitting IP packets, and an overlay control plane, namely the Resource Control Layer (RCL) that is based on the Bandwidth Broker (BB) concept. Although the classical BB architecture proposes a concentrated approach where one BB is responsible for an administrative domain, RCL is designed as a distributed BB, to overcome scalability problems. As depicted in Figure 2-1, the three key components of the RCL are: the Resource Control Agent (RCA), the Admission Control Agent (ACA) and the End-User Application Toolkit (EAT).
The *Resource Control Agent* represents the ultimate principle of the domain concerning the management of network resources. In order to simplify the task of the RCA to handle the resources efficiently, the network is

divided into sub-areas that form a tree structure, where each sub-area is initially assigned with its own resources, according to traffic load forecasts and results retrieved by measurements. The initial assignment of resources to sub-areas may not reflect the actual traffic load, so an intelligent load-balancing algorithm is utilized for the re-distribution of resources among them, as described in detail in [RPool]. The RCA is therefore divided in logical entities (Resource Pools, RP). Each RP manages the resources of the respective sub-area. The root of the tree is in charge of the available resources in the whole network, while each leaf of the tree structure (Resource Pool Leaf, RPL) is associated to one Admission Control Agent, which is in turn associated to one edge device of the network.

The *Admission Control Agent* mainly performs user authentication and authorization, reservation handling, and admission control. User authentication is initially performed when contacting the ACA, and then a reservation request, specifying the traffic requirements of the new flow, can be submitted. Subsequently, user authorization for that type of request is checked and a reservation state is created. Policing and admission control are only made at the edges of the network, therefore the corresponding ingress and egress points (ingress-egress ACAs) of the flow are identified and the local resources (in the RPL) are checked to ensure that the new flow can be accommodated. The core network is provisioned in order to ensure that once the admission control at the edges succeeds, no bottleneck will be created in the core network. Upon a successful reservation request, the corresponding ACAs consequently configure the edge routers appropriately to accommodate the new flow.

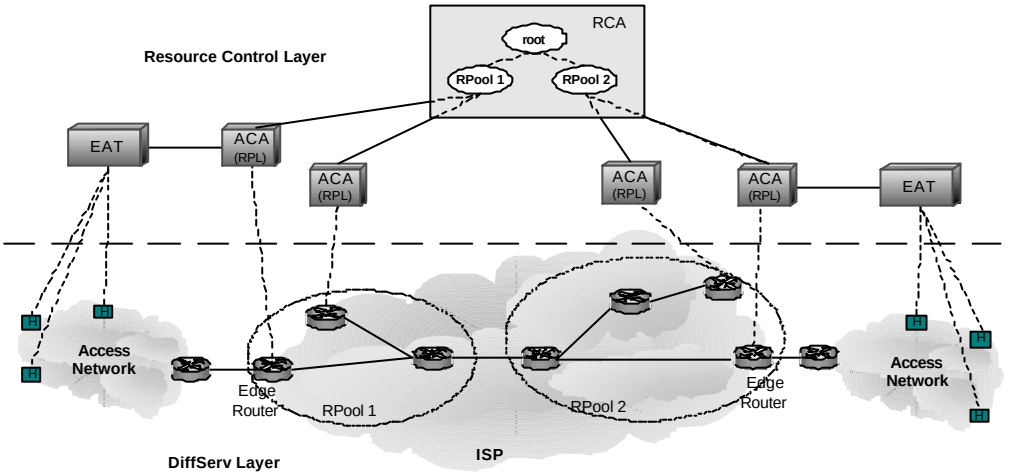


Figure 2-1: RCL structure and main interactions

Reservation requests are forwarded to the ACAs from the *End-User Application Toolkit*, which mediates between end-users or applications and the network. The EAT interacts with the ACA to become aware of the available network services. A reservation request specifies the flow identifiers, the selected network service and the traffic profile for it. Special support is foreseen for legacy applications as well as for end-users that are not aware of traffic description details, through the use of Application Profiles. Profiles are prepared through extensive testing of the application behaviour and stored in a repository. The EAT interprets the profiles and prepares the reservation on behalf of the user. A thorough description of EAT is included in section 3.

2.1 Network Services and Traffic Classes

The AQUILA architecture provides QoS guarantees to users by offering a number of transport options for user IP traffic. These are called Network Services, and are constructed by applying traffic conditioning to create aggregates that experience a known Per-Hop-Behaviour (PHB) at each node within the domain. Services can be clearly categorized as qualitative or quantitative depending on the type of the offered performance parameters. The AQUILA architecture supports five Network Services (NS), which are: Premium Constant Bit Rate (PCBR), Premium Variable Bit Rate (PVBR), Premium MultiMedia (PMM), Premium Mission Critical (PMC) and Best Effort (BE) [Net-Serv]. Applications can be grouped into this relatively small number of services, with the applications in each service having similar requirements on the network in order to perform effectively.

The PCBR network service is for constant and variable bit rate applications with low bandwidth flows. These applications have low delay and delay variation requirements, although are tolerant to some packet loss. In addition, they should have small packets so as not to provoke long transmission delays. IP telephony is the basic application supported by this NS. For good voice quality the delay should be less than 150msec for the 99% of in-profile packets and the packet loss should be less than 10^{-8}.
PVBR is appropriate for unresponsive VBR sources with medium to high bandwidth requirements. They have low delay, delay variation and packet loss requirements, even though less strict than those of PCBR. An end-to-end delay less than 250msec and a packet loss less than 10^{-4} are guaranteed. Video and teleconferencing are possible application supported by this NS.
PMM is expected to carry a mixture of TCP and non-TCP traffic. These flows require a minimum bandwidth, which must be delivered at a high probability. Independently of the transport protocol, flows are expected to implement some kind of congestion control mechanism and their aggressiveness should be similar to the one of TCP, assuming that they are roughly TCP-friendly. This NS delivers applications such as video/audio streaming and ftp. The drop probability for in-packets should be very low (less than 10^{-3}), while out-of-profile packets do not experience any QoS guarantees.
The PMC service supports mainly transactions and database queries. Thus, flows of the PMC are non-greedy, have short lifetimes, low bandwidth requirements and roughly homogeneous congestion control (TCP). The low bandwidth property allows overprovisioning of this service in the network, enabling minimal loss probabilities for in-profile packets and small queuing delays. Finally the BE requires no quality of service guarantees.
The implementation of the Network Services is realized with the use of network mechanisms, named the Traffic Classes (TCLs). A TCL is defined as a composition of a set of admission control rules, a set of traffic conditioning rules and a PHB. In AQUILA five TCLs are currently introduced: TCL1, TCL2, TCL3, TCL4 and TCL5 which correspond to PCBR, PVBR, PMM, PMC and BE, respectively. The scheduling mechanism selected is a combination of the Priority Queuing (PQ) and Weighted-Fair Queuing (WFQ) [PQ-WFQ]. A weight is assigned to each TCL, though a queue is dedicated for TCL1, which has strict priority over the other traffic classes. The rest TCLs are scheduled with the WFQ and each queue is managed by different queuing strategy (Drop-Tail, Random Early Detection (RED), Weighted-RED (WRED) [RED1, RED2]).

3 The End-user Application Toolkit

3.1 Functionality

Various architectures have been proposed for the provision of QoS support in the Internet. Among them, the AQUILA architecture proposes resource management and allocation mechanisms that promise efficiency and scalability. Also, a limited set of Network Services has been specified to cater for the diverse QoS requirements of different application types. An important issue in all QoS architectures is how those services are presented to the end-user, how the users may select the appropriate one and also how the actual provided QoS by the network may be confirmed. While a lot of effort has been focused on the specification of the mechanisms for QoS support in the core network, the issues of end-system support have been more neglected. More specifically, in the DiffServ/BB architecture, there is no standard way for applications and users to query the network of the offered service, to specify the desired levels of quality, to request QoS from the network and to verify it. Moreover, the task of selecting the appropriate QoS parameters for an application is a difficult task, even for application developers.
The EAT is a middleware architecture for the transparent support of QoS for applications and their users. The EAT provides efficient mechanisms for the presentation of the available QoS options to the user, for the selection of the appropriate quality parameters and the transmission of a reservation request to the RCL. It also helps users verify whether the requested level of QoS is honoured by the network. The EAT targets mainly legacy applications; that is, non-QoS aware applications that are widely used today but that cannot be modified to get QoS support. However, new applications may also be developed to use the functionality of the EAT, by using the QoS API the EAT provides.

The first step for the provision of QoS to an application is *QoS specification*. In the AQUILA architecture, a reservation request from the EAT to the corresponding ACA consists of a complex set of parameters that have to be filled by the user. This set includes:

- Flow specification: the expected traffic profile of the IP flow. Instead of using plain bandwidth, in the AQUILA architecture a more elaborate scheme, including token bucket specification, is used for multiplexing gain.
- QoS specification: the selected Network Service and the application requirements with regard to delay, delay variation (jitter) and packet loss.
- Filter specification: source and destination addresses of the IP flow. Also the TCP/UDP port numbers should be provided, as well as the protocol ID

A similar approach is adopted by the Integrated Services architecture. Obviously, such a request cannot be filled by any normal user that would like to make a reservation e.g. for his IP telephony session. Moreover, many application developers who would like to make their multimedia application QoS-aware by using a QoS API, would find it difficult to estimate the traffic profile of their application and the QoS it may need. The End-user Application Toolkit aims to help in this direction.

The EAT uses the *Application Profiles* that capture the QoS needs of a specific application. Being written in XML, Application Profiles can be easily parsed and serve the dynamic creation of the *Graphical User Interfaces* (GUIs). The *Converters* translate the high-level user choice into the network-level traffic and QoS specifications used by AQUILA and prepare a reservation request towards the RCL.

The resource request is also filled with the filter specification: the address and port numbers of the application flow. Especially the port numbers are often not known by the user. We use *Protocol Gateways* or *Proxies* to find this information.

The next step towards QoS support is *QoS negotiation and establishment*. The EAT handles the transmission of reservation requests to the RCL. The dynamically created Graphical User Interfaces are used to make such requests on behalf of legacy applications. However, new applications can directly use the *QoS API* provided by the EAT to make those requests.

QoS maintenance is the third step of QoS support. This includes monitoring of the level of service provided by the network and the adaptation of the reservation. With the use of the Proxy Framework, users are able to verify whether the agreed QoS levels match the actual traffic profile produced by the application, or if those levels are maintained by the network. In this way, users have the option to adapt the reservation to the changed traffic profile of the application. The Proxies incorporate basic measurement functionality: they can measure the traffic profile of a specific flow and they can estimate basic QoS parameters, like two-way delay.

The EAT is a distributed architecture: it consists of objects that can be physically located at different places in the network. All objects communicate with CORBA The EAT is not part of the core network, but resides in the access network of the end-user's host. The following figure illustrates the deployment scenario of the EAT components.

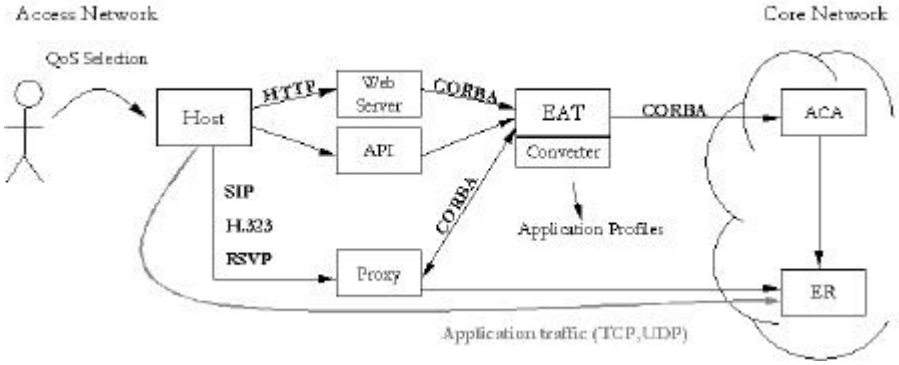


Figure 3-1: *The EAT deployment scenario*

3.2 Building Blocks

The following figure depicts the architecture, the basic building blocks of the EAT and shows the EAT middleware character between the applications, the end-user GUIs, and the RCL components.

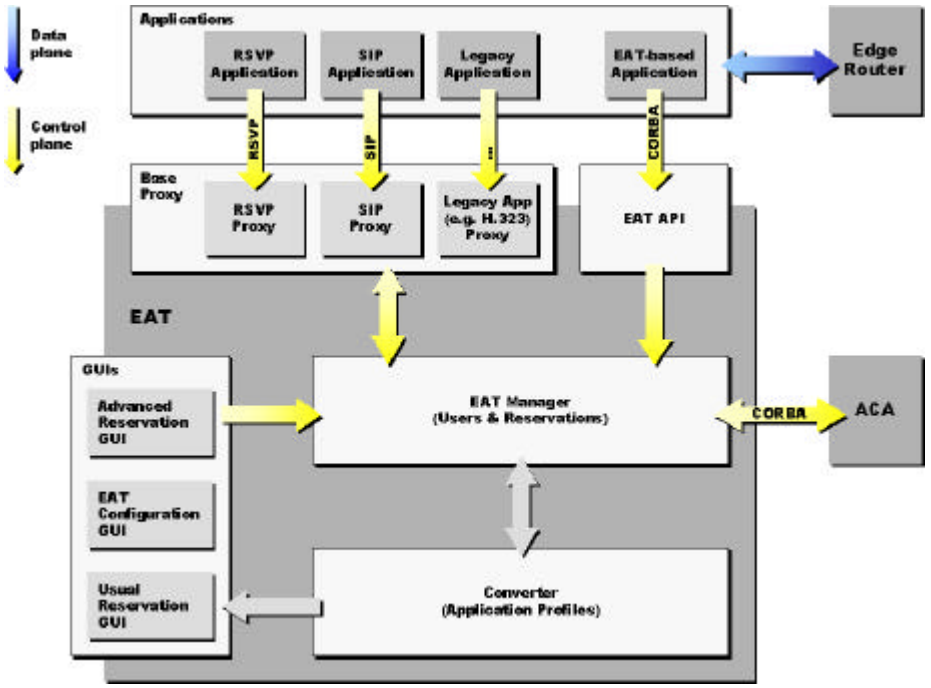


Figure 3-2: *The EAT's basic building blocks*

In the following, the EAT's components and their functionality are described in detail.

3.2.1 EAT Manager and APIs

The EAT Manager is the central component of the EAT realizing mainly user and reservation management. It acts as an intermediate component between the GUIs for reservation requests, the Converters, the Proxies of the EAT as well as the ACA. It has also access to the persistence layer in order to retrieve service information, subscriber data, and application profiles. In this way, it plays an important role in the described QoS user scenario of chapter 3.3, by handling most of the information *between* the EAT components and *towards* the RCL.
The EAT Manager provides its functionality for applications and end-user GUIs firstly via a CORBA interface, the so-called "internal" EAT API. This API provides the full functionality of the RCL concerning login, reservation requests and releases, network service retrieval, etc. The purpose of this API is to expose the EAT's features (such as the application profiles) to GUIs and applications.
On the top of this proprietary API, a "general" Java API is planned providing a standardised way for QoS request over multiple platforms and protocols (e.g. AQUILA, RSVP, WinSock2). This API aims to assist the developers of new, QoS-aware applications by offering high-level QoS interfaces for:

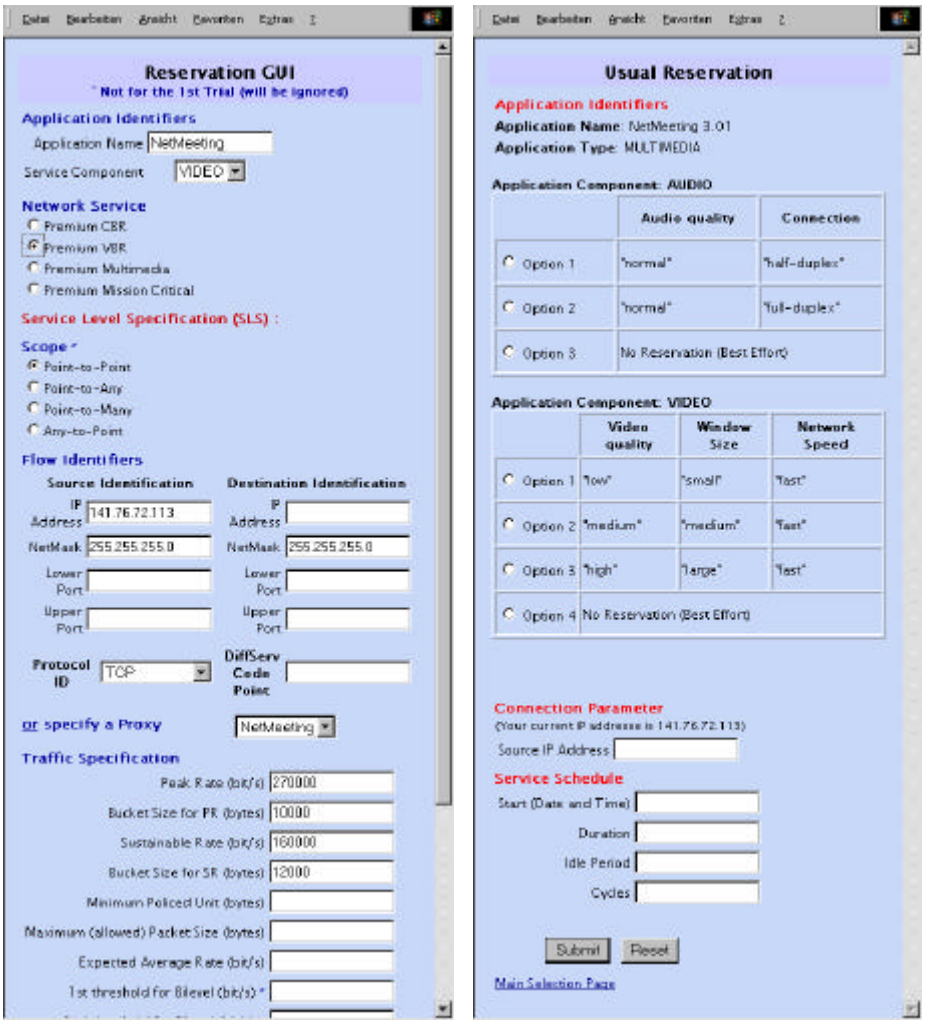
- Authentication
- Service retrieval and negotiation
- Establishment, modification and release of QoS requests
- Groups of single related reservation requests (e.g. for bi-directional services)
- Retrieval of accounting and measurements information

3.2.2 GUIs and Converters

The effective reservation, as mentioned in paragraph 3.1, takes place at the ACA through the delivery by the EAT Manager of a reservation request comprising, among other things, the network service and the SLS parameters. The easiest way to provide a QoS reservation GUI consists in offering a reservation request template, a mirror of the ACA reservation request, where the parameter values have to be filled (Figure 3-3-A). This approach is alas not very intuitional for a usual end-user.

In fact, a usual end-user is normally not a network specialist and therefore not able to handle the reservation request at network level. He wants to verbally express the QoS he perceives or requests at application level. At end-user level, QoS depends, on the human senses: sight, hearing, and on the perception of time-related behaviour. User-friendly descriptions for QoS correspond to a universal comprehension of applications and ideally make reference to well-known similar "services" from everyday life, like TV, VCR, hi-fi, or telephone, etc.

The Converter implements a mechanism – based on a mapping between end-user level, application level and network level – supporting the reservation process. This mechanism enables the construction of user friendly GUIs (Figure 3-3-B) where end-users can decide easily on a quality level for their current session.



A B

Figure 3-3: Graphical User Interfaces

A: QoS selection GUI without converting mechanism

B: User friendly QoS selection GUI after conversion, for legacy applications

The underlying concept for these mappings is the Application Profile. An Application Profile corresponds to a formal description of an application at end-user level as well as at network level. More specifically, the Application Profiles specify and store the high-level abstraction of the application's QoS requirements for the end-users, and also the estimated traffic specification of the application flows.

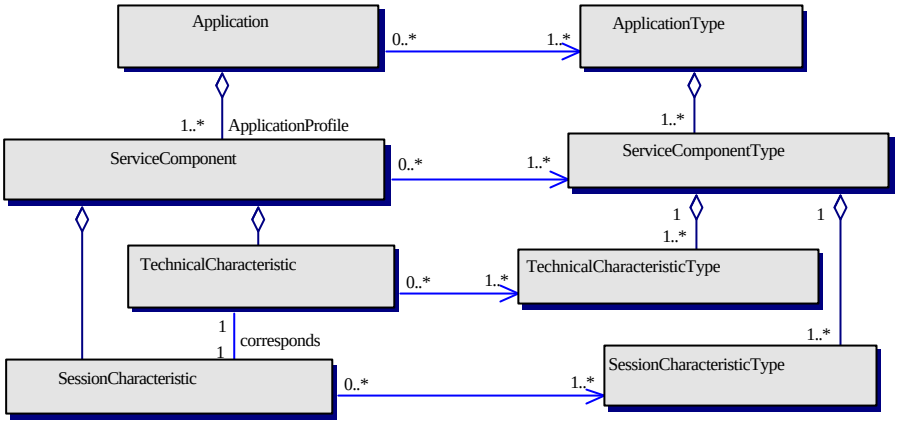


Figure 3-4: *Simplified UML class diagram of the description of an application. At a general or "meta" level, an Application can be associated to ApplicationTypes like "video conferencing", "voice over IP", "streaming media", etc. The "video conferencing" ApplicationType for example is described by the ServiceComponentTypes: "video", "voice" and "data". A ServiceComponentType itself is described on the one hand by network-oriented TechnicalCharacteristicTypes, e.g. the codecs, on the other hand by the user-oriented SessionCharacteristicTypes like "picture size", "sound quality", etc. A concrete application is described by a concrete Application Profile summarising the different possible QoS options and values for the concrete different ServiceComponents, TechnicalCharacteristics and SessionCharacteristics.*

In the sample case of a video-conferencing application, the concrete profile (in our case an XML file) contains the list of the service components with their quality options and technical sets of values. For example NetMeeting has an audio, a video, and a data service component. The video service component has three possible options (fixed by the application itself): "high quality", "medium quality", and "low quality". Each option has many session characteristics sets and their corresponding technical characteristic sets (see Figure 3-5).
The converter on the basis of the end-user information (login, SLAs, available QoS) and the session characteristics of the application profile prepares the user-friendly GUI and presents it to the end-user like in Figure 3-3-B. The corresponding technical characteristics parameters, whose values have been determined by measurements, remain transparent for the end-user. The Converter then maps (one to one mapping) and transforms the end-user selection into a reservation request using the corresponding technical characteristics values.

```
<ApplicationProfile name="NetMeeting" version="3.01" type="MULTIMEDIA">
...
   <ServiceComponent type="VIDEO">
      <description>There are three Options for Video</description>
...
         <Option optionID="Video_1" description="Video Low Quality scenario">
         <SessionCharacteristic>
            <SessionCharacteristicParameter>
               <name>Video quality</name>
               <description>perceived video quality</description>
               <qualifier>"low"</qualifier>
            </SessionCharacteristicParameter>
            <SessionCharacteristicParameter>
               <name>Window Size</name>
               <description>window size</description>
               <qualifier>"small"</qualifier>
```

```
            </SessionCharacteristicParameter>
            <SessionCharacteristicParameter>
               <name>Network Speed</name>
               <description>selected network speed</description>
               <qualifier>"fast"</qualifier>
            </SessionCharacteristicParameter>
         </SessionCharacteristic>
         <TechnicalCharacteristic>
            <serviceID value="PVBR"/>
            <TrafficSpec>
               <PR Unit="bit/s">270000</PR>
               <BSP Unit="bytes">10000</BSP>
               <SR Unit="bit/s">160000</SR>
               <BSS Unit="bytes">12000</BSS>
               ...
            </TrafficSpec>
         </TechnicalCharacteristic>
      </Option>
      <Option optionID="Video_2" description="Video Medium Quality scenario">
...
```

Figure 3-5: *Extract of the NetMeeting profile, 1st option of the video service component*

In this manner we are able to provide QoS to usual end-user in a user friendly way. This mechanism is particularly adapted for legacy applications, although it is not easy to determine the parameter values.

3.2.3 Proxy Framework

Applications, especially multimedia ones, usually open data connections with the use of connection setup protocols. Example protocols are the Session Initiation Protocol (SIP) and the H.323 protocol suite for the setup of IP telephony calls. As already mentioned, for the correct classification and marking of those user flows at the ingress of the core network, the source and destination IP addresses and TCP/UDP port numbers of the connection are required. However, the port numbers are always not well-known and usually are negotiated during the call setup.

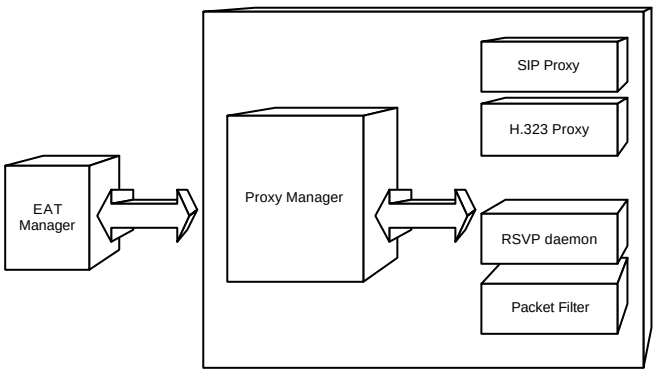


Figure 3-6: The Proxy Framework

The main goal of the Proxy Framework [Proxy] is to interfere in this protocol exchange, intercept and translate the messages, extract their content and pass it to the EAT Manager. The Proxy Framework consists of a set of Application-level Proxies or Protocol Gateways. There is one Protocol Gateway for each connection setup protocol. The Protocol Gateways are instantiated and controlled by a central entity, the Proxy Manager (PM). The Proxy Manager is also responsible for the communication with the EAT Manager. The internal structure of the Proxy Framework is depicted in the Figure 3-6.

As can be deduced from the figure, apart from the supported Protocol Gateways, two more modules comprise the full Proxy Framework: the Packet Filter and the RSVP daemon. Both modules reside at the egress of the user's access network, usually inside the edge router or the corresponding firewall. The Packet Filter operates in a complementary mode to the Protocol Gateways. Its main role is to filter all packets that cross the boundaries of the access network and perform measurements on selected IP flows. Such measurements include token bucket measurements that will assist the EAT in determining the correct parameters for a reservation, but also simple QoS measurements like throughput and two-way delay. The Packet Filter is also used for the detection of new flows with possible multimedia content, based on the RTP Payload header field.
The RSVP daemon module is used for the transparent support of RSVP-aware applications in the AQUILA architecture. In this way, RSVP can be used as another QoS signalling mechanism next to AQUILA's CORBA based signalling. Following the IntServ-DiffServ interoperability model [Int-Diff], the RSVP daemon forwards transparently the RSVP messages within the DiffServ core network and intercepts them at its edges. The content of the PATH and RESV messages is consequently passed to the EAT Manager for the establishment of reservation by the Resource Control Layer.

3.3 User scenario

In this paragraph we present a scenario that describes the succession of the EAT operations when a user requests QoS for his application.

1. **Login:** In a first step, the end-user logs in to the Resource Control Layer with his user name and password through the EAT web portal. The EAT Manager creates a new local user object and forwards the login data to the ACA in order to check whether the user is registered or not.
2. **QoS Offer:** There are two ways to offer QoS to the end-user: an advanced mode for specialists, and a usual mode for non-specialists:
    a. **Advanced mode:** The user gets a technical oriented and very detailed QoS reservation form that has to be filled. This form includes the network services offered by the AQUILA RCL. The user has to select the appropriate service and provides all the required parameters (Figure 3-3-A). Those parameters include the addresses and port numbers of the flow, its traffic profile as well as the requested QoS.
    b. **Usual mode:** In this mode the QoS is offered as pre-defined QoS options. First, all applications supported by the EAT are presented to the user. Second, after the selection of an application, the end-user gets the available QoS options (Figure 3-3-B). Eventually, the selected option is passed to the Converter, which is responsible for the preparation.
3. **QoS Request:** By clicking the "submit" button the reservation data are sent to the EAT Manager. The EAT Manager determines whether the request can be forwarded to the ACA immediately, or if additional information data are needed from the Proxy Framework: as discussed in paragraph 3.2.3, for some applications the TCP/UDP port numbers are not known in advance but they are negotiated during call setup. Two cases are foreseen:
    a. **No Proxy is needed:** A new local reservation object is created, and – if all necessary parameters are known – the *Reservation Request* can be sent to the ACA. The end-user is informed of the success of the request. He can start the application with QoS.
    b. **A Proxy is needed:** The reservation is marked as "provisional" and registered at the Proxy; the flow parameters are not known at that time. So the reservation cannot be requested at the ACA. The Proxy, after the start of the application (for example a NetMeeting conference session) is going to detect the missing flow parameters (e.g. the dynamically negotiated port numbers), and then sends them back to the EAT Manager. It can afterwards make a regular *Reservation Request* at the ACA. The user can now close his web browser and use the application with QoS.
4. **Reservation Release:** At the end of the application session, the end-user can release his reservation(s) by using a separate web page, and consult the accounting information (such as traffic volume, duration) of the QoS session. The release message is forwarded by the EAT Manager to the ACA in order to de-allocate the network resources.

5. **Logout.** As an option, the end-user can also logout from the EAT and the RCL. In this case all his active reservations are automatically released.

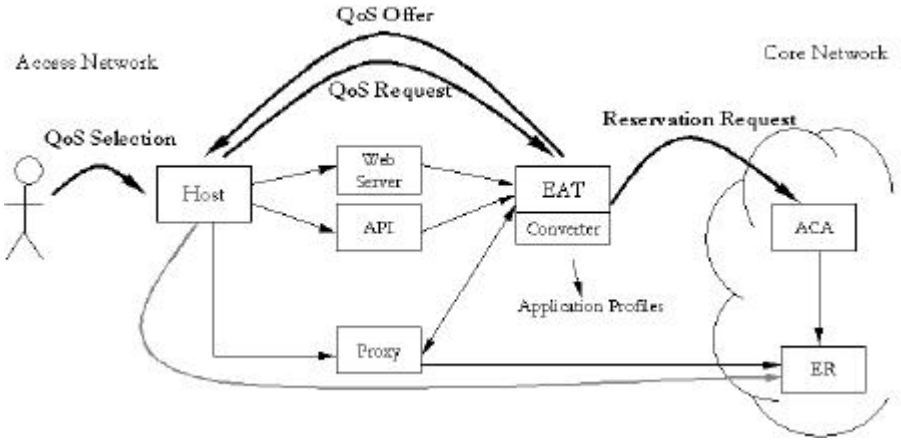


Figure 3-7: *QoS scenario between end-user, EAT, and ACA*

4 Implementation Issues

The RCL as well as the EAT itself have modular architectures with well-defined communication interfaces in between. These interfaces are specified in the programming language independent *Interface Definition Language (IDL)*, and the communication between the different components relies mainly on the *Common Object Broker Architecture (CORBA)*, the OMG's vendor-independent architecture and infrastructure that computer applications use to work together over networks. As an example, the Proxy as well as the EAT Manager specify methods for their asynchronous communication (see the scenario in paragraph 3.3), namely registerApplication() in proxy.idl and setNetworkInfo() in eatManager.idl.
The EAT as well as all other components of the RCL are implemented in the platform independent, object oriented programming language Java. Furthermore, modern Java technologies are used for the following purposes: The EAT's graphical user interfaces, i.e. the web pages, are realised by Java Server Pages (JSPs) [JSP] based on Java servlets [Servlets], due to their partly dynamically created content. Network configuration data as well as network services are stored in a central directory service accessible via the Lightweight Directory Access Protocol (LDAP). The application profiles are – due to the need for a flexible, hierarchical structure – specified in the eXtensible Markup Language (XML) [XML].
The use of JSPs and servlets in combination with the EAT API that offers QoS options and supports QoS requests, makes it possible to create complex internet services that integrate and bind existing well-known internet applications in a common QoS supporting portal. The main advantage of a web-based approach is the fact that no modifications to applications or to the operating system are necessary. No installation of QoS agents is needed on the end-user hosts; a web browser need only be used for the user to make QoS requests.

5 Conclusions and Outlook

In this paper we have presented a web-based architecture for the support of QoS for legacy applications. The End-user Application Toolkit (EAT) is a middleware residing at the access of the network that provides the mechanisms for the presentation of network services to users, for the selection of the appropriate service, as well as the supervision of the received quality. The EAT provides high level abstractions for QoS parameters, in an effort to make QoS selection by the users an easier task. The EAT focuses on legacy applications; popular currently used applications that cannot be modified to leverage the QoS capabilities of a QoS-enabled network. However, new applications can be created that use the EAT QoS API for QoS support.

The main strength of the EAT approach is the help that it provides to average users. Most users are not able to express their QoS requirements in quantitative terms. For instance, they cannot know what is the upper limit for tolerable packet delay or jitter in an IP telephony session. Moreover, they cannot provide the traffic specification of their application flow. What they can do is select a relative level of quality: good, medium or no quality. The EAT provides a high level abstraction of QoS in a way comprehensible to the normal user and maps the selected level into actual QoS parameters.
The web-based architecture of the EAT is another advantage of the AQUILA approach. Users can make reservations for their applications by using only their web browser. No special software is required on their hosts.
A result of this approach is that applications cannot be directly aware of QoS. Since they cannot be modified, they cannot make reservation requests by themselves or cannot be informed directly about the quality of service received. More important, the output of the applications, i.e. the traffic produced by an application cannot be influenced by the EAT, in order to comply with the agreed traffic profile. Contrary to the application adaptation approach where the application adapts its traffic according to network conditions, reservation adaptation is selected for the EAT: through the GUIs, users have an overall view of the network performance and can accordingly send a new reservation to the RCL.
The End-user Application Toolkit provides the most viable and efficient solution for QoS support for legacy applications. With little effort, many legacy applications can be supported by our architecture. This effort includes testing the application in order to find its fundamental traffic characteristics and developing the appropriate Application Profile. For some applications also, the implementation of a Protocol Gateway may also be required.

6 Acknowledgement

This work was performed in the framework of IST Project AQUILA (Adaptive Resource Control of QoS Using an IP-based Layered Architecture – IST-1999-10077) funded in part by the EU. The authors wish to express their gratitude to the other members of the AQUILA Consortium for valuable discussions.

REFERENCES

[AQUILA] *The IST AQUILA project home page, http://www-st.inf.tu-dresden.de/aquila/*

[BB] *K. Nichols et al., "A Two-bit Differentiated Services Architecture for the Internet", RFC 2638.*

[CORBA] *CORBA/IIOP Specification, http://www.omg.org/technology/documents/formal/corba_iiop.htm*

[DiffServ] *S. Blake et al., "An Architecture for Differentiated Services", RFC 2475.*

[IntServ] *R. Braden et al., "Integrated Services in the Internet Architecture: an Overview", RFC 1633.*

[Int-Diff] *Y. Bernet et al., "A Framework for Integrated Services Operation over Diffserv Networks", RFC 2998.*

[JSP] *Sun Microsystems, "JavaServer Pages", http://java.sun.com/products/jsp/*

[Net-Serv] *S. Salsano et al, "Definition and usage of SLSs in the AQUILA consortium", Internet Draft, November 2000, http://www.ietf.org/internet-drafts/draft-salsano-aquila-sls-00.txt*

[PQ-WFG] *J. Bennett, H Zhang, "Hierarchical packet fair queueing algorithms" SIGCOMM, August 1996, pg. 143-156.*

[Proxy] *Ch. Tsetsekas, S. Maniatis I.S. Venieris, "Supporting QoS for Legacy Applications", ICN '01, Colmar, France, July 2001.*

[RCL] *M. Winter et al., "System architecture and specification for the first trial", AQUILA deliverable D1201, Munich, Germany, June 2000.*

[RED1] *V. Firoiu, M. Borden, "A study of active queue management congestion control," Infocom 2000, Tel-Aviv, Israel, March 2000.*

[RED2] *T. Ziegler, C. Brandauer, S. Fdida, "A quantitative model for parameter setting of RED with TCP traffic", 9th International Workshop on Quality of Service, Karlsruhe, Germany, June 6-8, 2001.*

[RPool] *E. Nikolouzou, G. Politis, P. Sampatakos, I. Venieris, "An adaptive algorithm for resource management in a differentiated services network", IEEE ICC2001, Helsinki, Finland, June 2001.*

[RSVP] *R. Braden et al., "Resource ReSerVation Protocol (RSVP)", RFC 2205.*
[Servlets] *Sun Microsystems, "Java Servlet Technology", http://java.sun.com/products/servlet/*
[XML] *W3C, "Extensible Markup Language (XML)", http://www.w3.org/XML/*