International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:4, 2008

# QoS Management in the Future Internet

S. Rao, S. Khavtasi, C. Chassot, N. Van Wambeke, F. Armando, S. P. Romano, and T. Castaldi

***Abstract*—**The talks about technological convergence had been around for almost twenty years. Today Internet made it possible. And this is not only technical evolution. The way it changed our lives reflected in variety of applications, services and technologies used in day-to-day life. Such benefits imposed even more requirements on heterogeneous and unreliable IP networks.

Current paper outlines QoS management system developed in the NetQoS [1] project. It describes an overall architecture of management system for heterogeneous networks and proposes automated multi-layer QoS management. Paper focuses on the structure of the most crucial modules of the system that enable autonomous and multi-layer provisioning and dynamic adaptation.

***Keywords*—**Automated QoS management, multi-layer provisioning and adaptation, QoS, QoE.

## I. Introduction

THE incremental and any-purpose usage of Internet resulted in uncontrollable and degraded performance, revealing need for reasonable utilization of network resources. The more heterogeneous and complex the network becomes the more sophisticated solutions it requires to satisfy the QoS. Network operators are facing great challenges of diversified and heterogeneous structure of Future Internet in attempts to provide capable QoS mechanisms and technologies.

The solution can be found in the open and flexible system which would reflect preferences of all participants. Moreover the system should be automated and responsive to changes by dynamically adjusting the network parameters in case of violations.

The main requirements to the architecture of such system would be:

- System heterogeneity and complexity should be transparent to user and all preferences have to be effectively expressed and stored.
- It should be as ubiquitous and modular as possible
- It should possess easily extensible and pluggable structure.

Manuscript received March 30, 2008. The NetQoS Project has received funding from European Commission.

S. Rao and S. Khavtasi are with Telscom AG, Sandrainstrasse 17, Bern 3007, Switzerland (e-mail: {rao, sophia}@telscom.ch).

C. Chassot, N. Van Wambeke and F. Armando are with LAAS/CNRS; Université de Toulouse ; 7 Avenue du Colonel Roche, F-31077 Toulouse, France (e-mail: {chassot, van.wambeke,armando}@laas.fr).

S. P.Romano and T. Castaldi are with Universita' di Napoli Federico II Computer Science Department (e-mail: {spromano, tobia.castaldi@unina.it).

The main requirements to the functioning of such management system can be formulated as follows:

- Provision of necessary QoS to a particular applications based on the actors policies
- Dynamic adaptation in case of violations based on actors' preferences and/or policies.

## II. Beyond the State-of–the-Art

There has been extensive research and development on QoS support within each layer of ISO stack. In the context of complexity of modern networks it often appears that one particular QoS technique is not sufficient for the desired QoE. Moreover, the conventional solutions are not flexible and responsive to changes – therefore they are not dynamic and adaptive. The NetQoS project advances the state-of-the art by bringing together multi-layer QoS approach and automated QoS provisioning and adaptation.

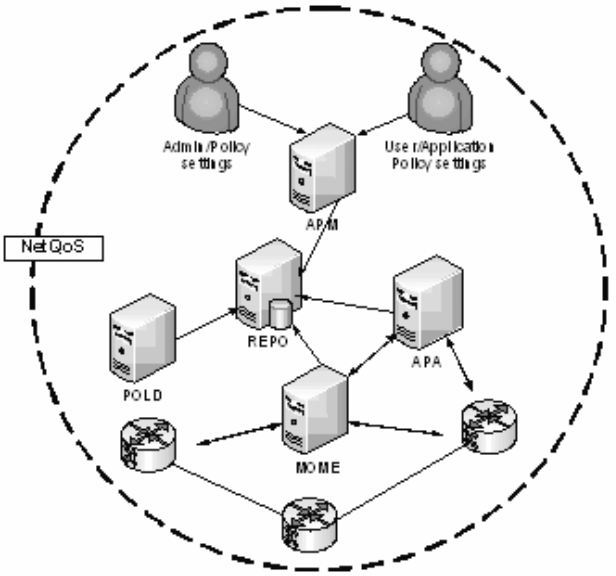


Fig. 1 Architecture of NetQoS system

Fig. 1 illustrates simplified NetQoS architecture with its main modules: Actor Preference Manager (APM), Monitoring and Measurement (MOME), Policy Description Manager (POLD), Policy repository (REPO) and Automated Policy Adaptor (APA).

APA acts as the central entity of the system. It does not provide QoS by itself, but is aimed at providing and dispatching operational network and/or transport level policies

that help the QoS-oriented communication system to take into account the dynamic policies expressed by actors. The APA derives a set of operational policies, enforces them and updates them as needed when the system evolves. MOME contains all the monitoring and measurement activities associated with the management of the context evolution (actors policies evolving, end system/network resource changing), the evaluation of the operational policy efficiency and the reporting of quality information.

POLD represents a set of ontologies used to specify the actors' level policies, the operational policies, etc. The APM is aimed at providing service to users (or external actors) with a GUI/API allowing them to define actor-level dynamic policies for the system based on ontologies. These policies (requirements, preferences, profile, quality reporting) may be expressed before or during the communication with system and are stored in REPO [2].

The NetQoS system targets autonomous policy-based QoS management for heterogeneous networks. As such, it is necessary that the components are capable to autonomously manage the network by handling actors' requests and reacting to all possible events. Hence, the need arises of a component that coordinates all the others. In the NetQoS architecture such a coordinating component is denoted as the Context Manager (CM) – part of MOME module. The Context Manager is in charge of identifying the "context", i.e. the operational status of the system and all the relevant events. In case an event that requests actions from other components occurs, the Context Manager must become aware of it and notify the appropriate components.

Depending on the event, different modules of NetQoS system are animated. Fig. 2 illustrates working principle of CM.

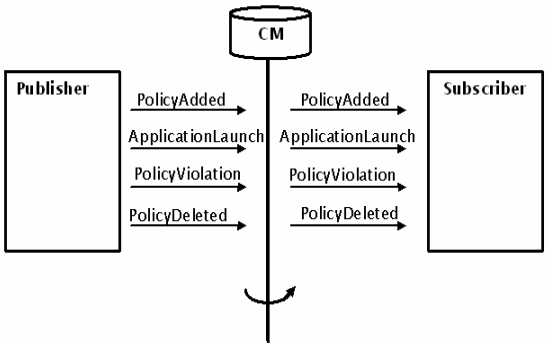


Fig. 2 Working principle of CM

Therefore, the Context manager acts as the main Publication/Subscription mechanism of the system. It represents a mediator which is spinning the events from Publisher to Subscriber, where Publisher/Subscriber can be any module of the system.

The examples of events that the Context Manager has to be aware of are the launch of an application, the violation of a policy, the addition of new policies, etc. Under this perspective the CM helps the system to become autonomic, since it provides it with the following properties:

- Automatic: this essentially means being able to self-control its internal functions and operations. An autonomic system must in fact be self-contained and able to operate without any external intervention
- Adaptive: an autonomic system must be able to change its operation. This allows the system to cope with temporal and spatial changes in its operational context
- Aware: an autonomic system must be able to monitor its operational context as well as its internal state in order to be able to assess if its current operation serves its purpose. Awareness controls adaptation of its operational behavior in response to situation or state changes.

The CM is a crucial component in the overall NetQoS architecture, since it allows correct operation of the framework through proper connection of the various components taking part in the dynamic network configuration/adaptation cycle. The CM makes it possible to let such interactions be based on an asynchronous paradigm, thus improving the flexibility of the NetQoS solution and providing good scalability.

## III. Multi-Layer Provisioning and Adaptation

The objective of the policy-based networking paradigm is to enable a formulation of operational goals on the level that is more generic and abstract than the low level configurations understood by networking devices. These operational goals are often related to the performance indicators of the network. The autonomous management dictates that the system must be able to automatically evaluate whether the performance goals – as specified in policies – are met. It is therefore required to map these performance related policies into specifications for monitoring and measurement activities that can then be executed by the MOME. Among other reasons, the violation of performance goals should trigger an adaptation of the autonomous system. This requires a component that is capable of dynamically adapting policies: the Automated Policy Adaptor (APA).

The APA acts as the central engine which performs the automated policy adaptation process, i.e. to define, adapt and enforce the operational policies at one or several layers of the QoS stack (e.g. Network or Transport layers). These policies define the services to be composed with the aim to satisfy the policy objectives expressed in terms of QoS goals (delay, bandwidth, etc).

These functions are performed by the Policy Decision Manager (PDM), the Policy Adaptation Manager (PAM) and the Policy Enforcement Manager (PEM) (Fig.3):

- PDM's decisions act on the services architecture to be applied by the managed entities; for instance, to satisfy the QoS goals for a given application, which are supposed to be expressed in terms of variable delay and partial reliability, the PDM may decide to select a partially reliable service at the Transport level on top of

an AF-based differentiated service at the IP level.

- PAM's decisions act on the value of the parameters of the services selected by the PDM; for instance, the PAM may decide to modify the value of the loss rate to be reached by the partially reliable Transport service, depending on the effective QoS provided by the AF-based IP service.
- PEM is in charge of dispatching these decisions toward Agents that allow interfacing the APA with the managed entities, typically border routers at the Network level or Transport entities on the end hosts at the Transport level. The goal of the Agent is to convert PDM and PAM generic decisions into specific decisions that depend on the managed entities [3].

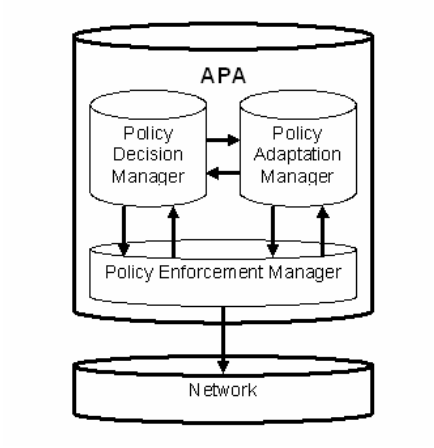


Fig. 3 Architecture of APA

*A. PDM*

The class diagram of the PDM is given in Fig. 4.

Basically, the PDM is divided into three active parts that it creates when it is launched (i.e. at the start of the system).

- *PDMNotificationManager* processes the notification events coming from the CM and retrieves the corresponding policy(ies) from the REPO;
- *PDMWorkerManager* processes the needed provisioning decision at one or several levels (with the help of the adequate *PolicyDecisioners*);
- *PDMDispatcher* processes the interactions with the PEM(s) to make it enforced the policy, and stores it in a local repository (*Op_REPO*). It also sets to Active the policy in the REPO.

*PDM Internal Components*

***PDMNotificationManager.*** The *PDMNotificationManager* processes the notification events for which it has subscribed to (e.g. launch of a new application). Whenever a notification event occurs, it gets the corresponding policy(ies) to be processed, stores it(them) in a local database and invokes the *processDecision(Policy)* method of the *PDMWorkerManager*.

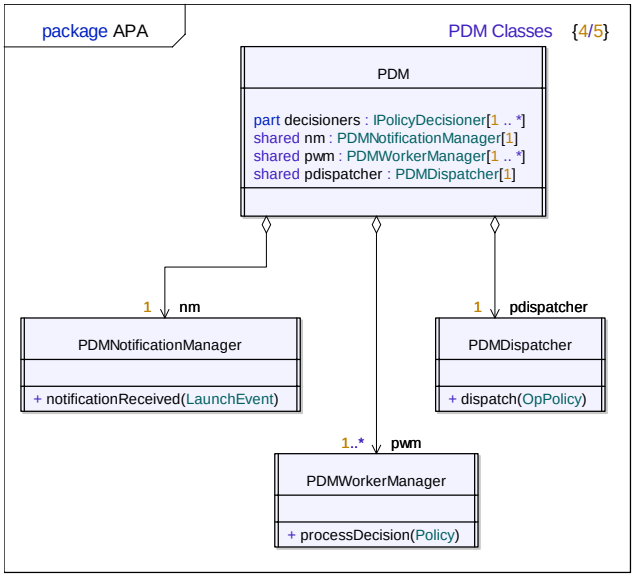


Fig. 4 Class Diagram of the PDM internal components

*PDMWorkerManager*. When it is asked to process a decision, the *PDMWorkerManager* invokes the *takeDecision(OpPolicy) : OpPolicy* method of the adequate *Decisioner(s)*, either at a single level or at several levels successively from the lowest to the highest one.

To perform this task, the *PDMWorkerManager* makes use and coordinates one or several *PDMWorkers*, whose number may be defined depending on the criteria such as the maximum number of policies to be processed in the same period. This distribution of the work is aimed at allowing the test of different configurations in front of scalability issues related (for instance) to the number of application launches that could occur during the same period. Once retrieved the operational policy to be applied at one or several levels, a *PDMWorker* invokes the *dispatch(OpPolicy)* method of the *PDMDispatcher* (described here after).

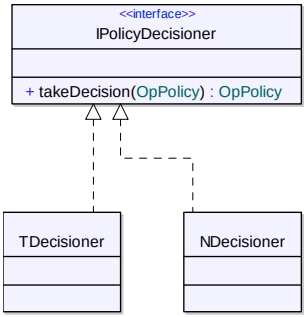


Fig. 5 Class diagram of the Decisioners at Network and Transport levels

***Decisioner[i].*** Decisioner contains the provisioning logic for the level i. Its API offers a single method (called takeDecicion(OpPolicy): OpPolicy) which returns the operational policy to be applied at the considered level i. As

an input, this method takes the operational policy that has been defined at level i-1.

As an example, next Fig. 5 illustrates the instantiation of two Decisioners, one at the Network level, one at the Transport level.

*PDMDispatcher*. When it is invoked by a PDMWorker, the PDMDispacher invokes the *enforcePolicy(OpPolicy)* method of the PEM, then it stores the operational policy in a local repository (Op_REPO). It also sets to active the policy in the REPO.

### *B. PAM*

The class diagram of the PAM is given on Fig. 6.

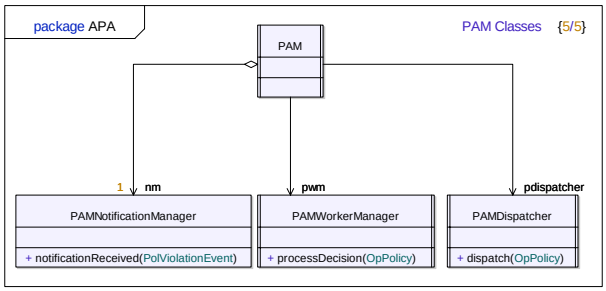


Fig. 6 Class Diagram of the PAM internal components

Similarly to the PDM, the PAM is basically divided into three active parts that it creates when it is launched (i.e. at the start of the system):

- *PAMNotificationManager* processes the notification events coming from the CM and retrieves the corresponding (intermediate) policy(ies) and operational policies;
- *PAMWorkerManager* processes the needed adaptation decisions at one or several levels (with the help of the adequate PolicyAdapters);
- *PAMDispacher* processes the interactions with the PEM(s) and makes it to enforce the adapted policy.

*PAM Internal Components*

***PAMNotificationManager.*** The *PAMNotificationManager* processes the notification events for which it has subscribed (e.g. a policy violation). Whenever a notification event occurs, it gets the corresponding intermediate policy(ies) and operational policy to be considered and adapted, stores them in a local database and invokes the *processDecision(OpPolicy)* method of the *PAMWorkerManager*.

***PAMWorkerManager.*** When it is asked to process an adaptation, the *PAMWorkerManager* invokes the *adaptPolicy(OpPolicy) : boolean* method of the adequate *Adapter(s)*, either at a single level or at several levels successively from the lowest to the highest one.

Similarly to the *PDMWorkerManager*, to perform this task, the *PAMWorkerManager* makes use and coordinates one or several *PAMWorkers*, whose number may be defined depending on the criteria such as the maximum number of policies to be adapted in the same period. If the result of the adaptation is positive, the PAMWorkerManager invokes the dispatch(OpPolicy) method of the PAMDispatcher. If the adaptation is not possible, the PDM is informed by an alarm that a re-provisioning as to be done.

***Adapter[i].*** Adapter[i] contains the adaptation logic for level i. Its API offers a single method (called *adaptPolicy(OpPolicy) : boolean*) which returns if the adaptation has succeeded.

As an example, next Fig. 7 illustrates the instantiation of two *Decisioners*, one at the Network level, one at the Transport level.

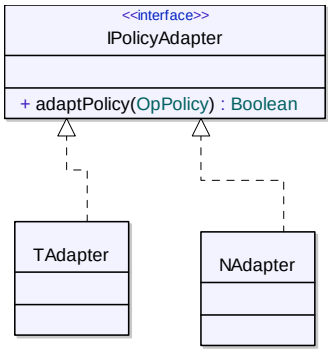


Fig. 7 Class diagram of the Decisioners at Network and Transport levels

***PAMDispatcher.*** When it is invoked by a *PAMWorker*, the *PAMDispacher* invokes the *enforce(OpPolicy)* method of the PEM, then it stores the operational policy in a local repository (Op_REPO).

### *C. PEM*

The class diagram of the PEM is given on Figure 8. The Policy Enforcement Manager is responsible for contacting the required *Agent* component once the operational policy to be enforced has been defined by the PDM or PAM.

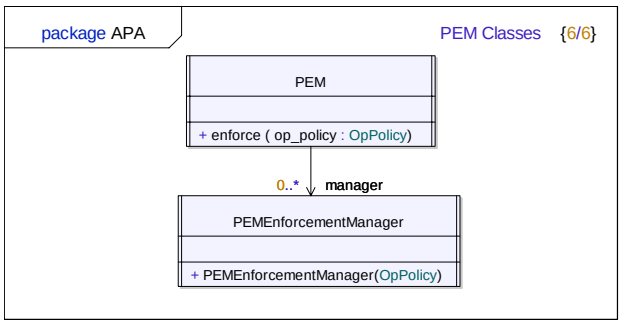


Fig. 8 Class Diagram of the PAM internal components

For each operational policy that is received by the PEM, an instance of the *PEMEnforcementManager* is constructed. This instance is then responsible for handling the request. By using active classes for these instances, multiple policies enforcements can take plan concurrently.

*PEM Internal Components*

In order to perform the enforcement, the *PEMEnforcementManager* instances that are created by the PEM for each policy to be enforced are detailed on the class diagram presented on Fig. 9.

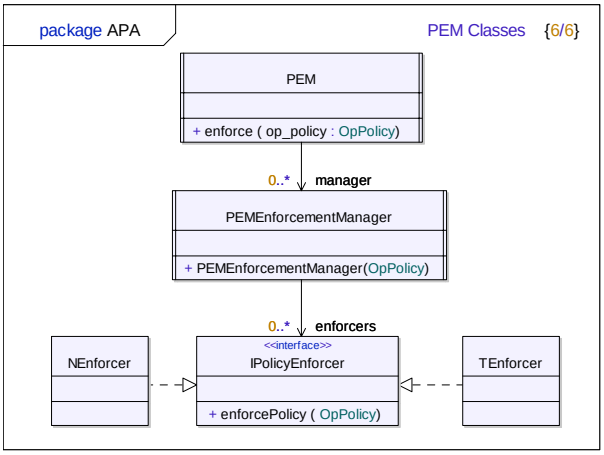


Fig. 9 Class Diagram of the PEM internal components

A *PEMEnforcementManager* is an internal component of the PEM that will, depending on the different policy levels included in the operational policy to be deployed, delegate the enforcement of the policy for each of these levels to the corresponding *IPolicyEnforcer* instance.

For example, an operational policy that contains information to be enforced both at the Transport and Network layer will be handled to both the instance of *IPolicyEnforcer* specialized in Transport Enforcement (*TEnforcer*) and Network Enforcement (*NEnforcer*).

## IV. NetQoS Management Example

The multi-layered approach towards QoS management can be demonstrated against two scenarios (illustrated on Fig. 10) in which:

- first one deals with the policy provisioning after an application launch;
- second one deals with a policy adaptation when the policy violation occurs [4].

For this purpose the APA is subscribed for 2 events from CM: *Application Launch* and *Policy Violation*.

*1) Scenario 1: Policy provisioning*

This scenario describes sequence of actions occurring in the system to meet the preferences of the actors. The interaction chain in this case is as follows:

1) APA subscribes to an Application Launch event via CM.
2) The application is launched.
3) The application launch is detected by the Traffic Identification Engine (TIE) of the MOME.
4) TIE informs the APA of this event, providing it with information related to the application type (e.g. VoIP, VoD) and connection identifiers (destination and source port numbers, IP addresses, etc).
5) Using this information, the APA retrieves (from the REPO) the intermediate policies (i.e. QoS goals and priority) associated to this application.
6) The APA elaborates the operational policy to be applied at the Network (resp. Transport level), and enforces it at the managed Network (resp. Transport) entities, via the Network (resp. Transport) Agents.
7) APA informs the MOME of the activation of a new operational policy for the identified connection. It also provides the MOME with the success criteria to be monitored. Finally, it sets to "active" the intermediate policy stored in the REPO.
8) The MOME installs the necessary elements on the involved hosts, in order to monitor the success criteria associated with the activated policy.

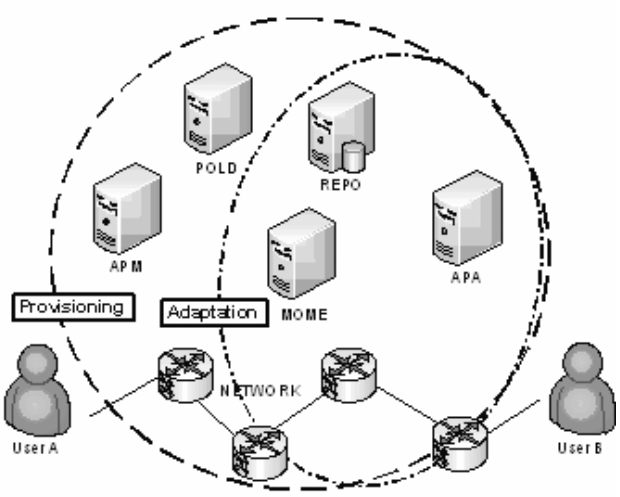


Fig. 10 Two scenarios for QoS management: provisioning and adaptation

*2) Scenario 2: Policy adaptation driven by a policy violation*

This scenario shows the sequence of actions that occur in case when the policy violation is observed and immediate reaction of system is required.

The interaction flow is as follows:

1) Once a policy violation for an active policy has been observed by the MOME tools, the MOME informs the APA about the event (via CM).
2) Using this information, the APA retrieves from the REPO the intermediate policies (i.e. QoS goals and priority) associated to this application.
3) The APA elaborates the adaptation to be applied at the Network (resp. Transport level), and makes enforces it at the managed Network (resp. Transport) entities via Network (resp. Transport) Agent.

4) APA informs the MOME (via CM) of the activation of a new operational policy for a given and identified connection. It also provides the MOME with the new success criteria to be monitored.

In the NetQoS example multi-layer APA consists from DataLink, Network and Transport layer Adaptors interconnected with the corresponding Agents in the network as illustrated on Fig. 11.

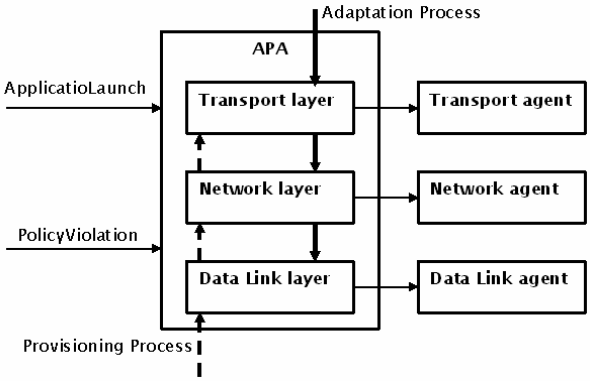


Fig. 11 Simplified functioning schema of APA

The Data-Link, Network and Transport layer provisioning and adaptation techniques considered and implemented in NetQoS are summarized below and describe state-of–the–art techniques used for QoS management.

### *A. Provisioning and Adaptation at the Transport Level*

For transport layer provisioning/adaptation NetQoS has investigated Enhanced Transport Protocol (ETP) [5], [6] as the modular approach and effective way to satisfy a wide variety of applicable requirements by composing and fine-tuning different well identified building blocks (rate control, shaping, congestion control, flow control).

The ETP combines both, the strengths of hierarchical and non-hierarchical frameworks, and proposes hybrid approach where a hierarchical plane is defined to implement QoS control functions and a non-hierarchical plane aimed at performing QoS management functions. In other words, per packet actions are performed following a hierarchical model while operations related to the management of the connection follow the event driven paradigm. This approach clearly separates the different planes that contribute in providing a given transport's service.

Given the highly demanding applications that evolve on heterogeneous networks (multimedia, VoIP etc) the transport service must be able to render a service that totally satisfies the applications' requirements. The modular approach of ETP seems to fulfil these criteria.

### *B. Provisioning and Adaptation at the Network Level*

The provisioning and adaptation rules on the Network level are stipulated by the recommendation of ITU G10.10 [7], where the main criteria like loss and delay parameters are specified.

Considering scalability issues the NetQoS focused on Differentiated Services approach, where the provisioning and adaptation on the network level can be done by exploiting the Differentiated Services Code Point (DSCP) marking. The advantages of such marking allow specifying high, medium and low loss priority for different applications, while the adoption of different queues allows specifying different delay metrics. Simplest implementation of such solution requires Linux IPtables and traffic control (TC) utilities.

Note that in the case of network provisioning/adaptation network devices remain transparent for APA. It just sends a generic request to Agent which is responsible to translate the request into format acceptable for particular router.

### *C. Provisioning and Adaptation at the Data Link Level*

The provisioning and adaptation rules at the Data link level aim to guarantee the behavior defined at the network level, where packets are marked with different code-points. The behavior corresponding to such code-point needs to be mapped onto the mechanisms made available by the particular data link technology. The rules at the data link level thus aim to translate the behavior defined by network level policies depending on the mechanisms available at the data link level.

In the context of 802.11 wireless networks, the challenge is to properly configure the Wireless Termination Point (WTP) agents residing in the devices holding the radio transceiver. Assuming the case when implementation of the WMM (Wireless MultiMedia) extensions is in the WTPs hardware, four different classes of services can be defined: Voice, Video, Best Effort and Background.

The behaviour associated with each class of service can be specified by means of four parameters: the Arbitration InterFrame Spaces (AIFS), the Minimum and the Maximum Contention Windows (CWmin and CWmax, respectively) and the Transmission Opportunity Limits (TXOPlimit).

Provisioning rules can be determined on the basis of a template approach. A rule is necessary to translate a specific code-point at the network level into a corresponding class of service at the data link level. The rules defining the four parameters of each service class may also be based on a template.

Adaptation rules are needed in case a violation of QoS guarantees occurs. Reacting to such a failure may consist of two steps. In a first step, a new rule can be defined to associate the flow(s) involved with a different service class. In case such a countermeasure fails and a new violation occurs, another new policy may be necessary to modify the parameters associated with the service classes. Such a rule may apply either to selected nodes crossed by the flow(s) involved or to all the nodes along the path followed by the flow(s).

## V. Conclusion

The concept presented in this paper aims at providing automated QoS management in heterogeneous networks. The investigated and implemented approach is outlined by the following features:

- Multi-layer provisioning/adaptation possibility which makes it flexible and ubiquitous solution.
- Smooth delivery of service due to dynamic adaptation.
- Improved user experience and enhanced end-to-end QoS.
- Platform is modular and open – providing the possibility to add other provision/adaptation modules through publishing/subscribing mechanism.

The process of provisioning/adaptation is not self-centric and involves participation of other modules of NetQoS system. Further information can be found on the project website [1].

## Acknowledgment

This paper reflects the ongoing progress of the project which is the result of collaboration of different organizations, therefore special thanks are addressed to NetQoS partners and this paper is based on the deliverables produced during work cycle of the project.

## References

[1] Project NetQoS website: http://www.netqos.eu.
[2] NetQoS project deliverable 2.2 NetQoS functional architecture
[3] NetQoS project deliverable 3.2 NetQoS system integration
[4] NetQoS project deliverable 2.5 Policy Implementation
[5] Nicolas Van Wambeke, Francois Armando, Christophe Chassot, Ernesto Exposito, "*A model-based approach for self-adaptive Transport protocols*", In Press, Elsevier Computer Communication's Special Issue on End-to-end Support over Heterogeneous Wired-Wireless Networks, 2008. doi:10.1016/j.comcom.2008.02.026.
[6] Christophe Chassot, K.arim Guennoun, Khalil Drira, François Armando, Ernesto Exposito and André Lozes, "*Towards Autonomous Management of QoS through Model-Driven Adaptability in Communication-Centric Systems*", International Transactions on Systems Science and Applications, Volume 2 Number 3 (2006), p. 255-264.
[7] ITU recommendation G.1010 available for download: ftp://ftp.tiaonline.org/TR30/TR303/Public/0312%20Lake%20Buena%20Vista/G1010%20-%2011-01.doc