On Policy-based Extensible Hierarchical Network Management in QoS-enabled IP Networks

P. Flegkas, P. Trimintzios, G. Pavlou, I. Andrikopoulos, C.F. Cavalcanti
Centre for Communication Systems Research
School of Electronic Engineering and Information Technology
University of Surrey, Guildford, Surrey GU2 7XH,UK
{P.Flegkas, P.Trimintzios, G.Pavlou, I.Andrikopoulos, F.Cavalcanti}@eim.surrey.ac.uk

Abstract

Policy-based Management has been the subject of extensive research over the last decade. More recently, the IETF has been investigating Policy-based Networking as a means for managing IP-based multi-service networks with quality of service guarantees. Policies are seen as a way to guide the behaviour of a network or distributed system through high-level, declarative directives. We mainly view policies as a means of extending the logic of a management system at runtime, so that it can be adaptive to changing or newly emerging requirements. We are interested in particular in the coexistence of "hard-wired" hierarchical management systems with policy logic in a fashion that the overall system becomes programmable and extensible. In this paper we consider generic issues behind hierarchical policy-based management systems and we present initial work on such a system for dimensioning and dynamic resource management in IP Differentiated Services networks.

Keywords

Policy-based Networking, Hierarchical Management, IP Differentiated Services

1. Introduction

Policy-based Management has been the subject of extensive research over the last decade [Moff93][Slom94]. More recently, the IETF has been investigating Policy-based Networking as a means for managing IP-based multi-service networks with quality of service guarantees [Stev99]. Policies are seen as a way to guide the behaviour of a network or distributed system through high-level, declarative directives. Although the declarative high-level aspect of policies is very important, particularly for human managers, we mainly view policies as a means of extending the logic of a management system at runtime, so that it can be adaptive to changing or newly emerging requirements. We are interested in particular in the coexistence of hierarchical management systems realised through "hard-wired" management logic with interpreted policy logic, targetting a programmable and extensible overall system. In this paper we consider generic issues behind hierarchical policy-based management systems and we present initial work on such a system for dimensioning and dynamic resource management of IP Differentiated Services networks.

The rest of the paper has the following structure. In section 2, we present an overview of the current state-of-the-art in policy-based research. This presentation is twofold, covering work in the research community, which includes policy classification, policy refinement and policy languages, and also work in the IETF, which includes recent work of the Resource Allocation and Policy Framework working groups. In section 3, we first discuss aspects of policy-based management and compare and contrast the approach to traditional, "hard-wired" management approaches. We then examine key aspects of hierarchical management systems and present a preliminary framework for the coexistence of such systems with management policies. In section 4, we present our preliminary work on hierarchical policy-based management for IP Differentiated Services, including an overview of the functional architecture for a relevant system and paying particular attention to policy aspects. We finally present a summary and point to our future research work in this area, in section 5.

2. Policy Management and Policy-based Networking State-of-the-Art

2.1 Policy-based Management in the Research Community

A lot of research has been carried out in the area of policies for the management of distributed systems, with most of the concepts pioneered by Imperial College London. [Slom94] presents the concepts of domains and policies in the context of a generic management architecture, see Figure 1. Assuming a distributed management system that reflects the distribution of the system being managed, policies are specified as objects which define a relationship between *subjects* (managers) and *targets* (managed objects). Policies are separated from the automated managers, facilitating the dynamic change of the behaviour and the adaptivity to new requirements without re-implementing the management applications. Domains provide the framework for partitioning management responsibilities by grouping objects in order to specify a management policy that applies to a domain. Domains are defined as objects, which maintain a list of references to their member managed objects [Slom89]. Figure 1 depicts a generic management architecture for distributed management systems consisting of: a) *communication services* for communication between applications as well as between applications and managed objects, b) *object services* for class and instance administration - the architecture assumes a distributed objects framework such as e.g. OMG CORBA, c) *distributed processing services* supporting interaction between distributed components, d) *common management services* which support the fundamental concepts of domains, policies and monitoring and e) the *management applications* which are capable of interpreting and applying policies.

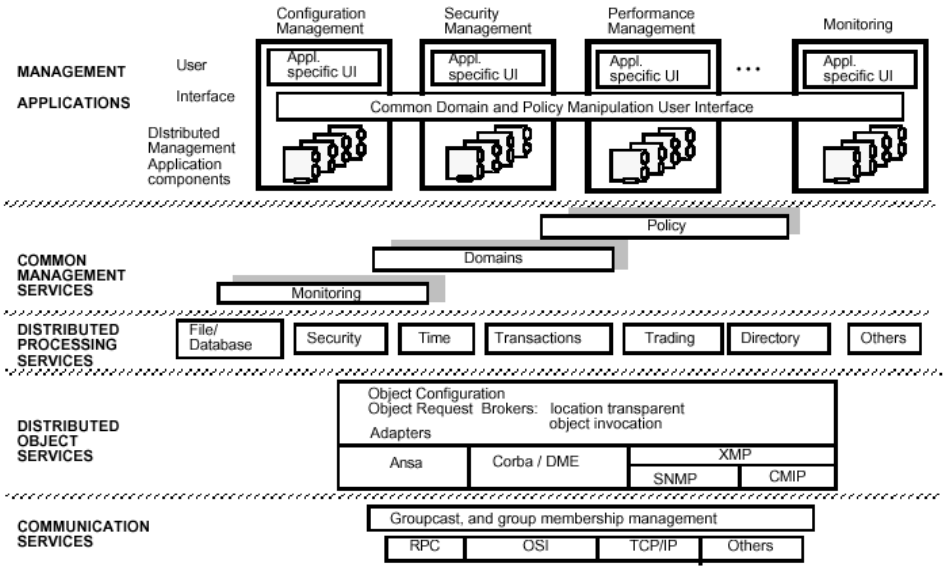

Figure 1 Policy-based Distributed Management System Architecture [Slom94].

The following types of policies are identified in [ICpol], while [Wies94] specifies also a list of criteria for the classification of policies:

Authorisation Policies (positive and negative), which specify what a subject is authorised/forbidden to do with respect to a set of managed objects. These are essentially access control policies.

Obligation Policies, which specify what operations the subject *must* perform on a set of target objects. Positive obligation policies are triggered by events.

Refrain Policies, which define the actions that subjects must not perform on target objects.

Delegation Policies (positive and negative), which specify which actions subjects are allowed to delegate to others.

[Moff93] discusses the issue of the refinement of a high level policy into a number of more specific lower level policies to form a policy hierarchy. Several different relationships can be identified between policies in a hierarchy:

Partitioned Targets: The target set of a lower level policy may be a subset of the target set of the higher-level policy.

Goal Refinement: the goal of a higher-level policy may be refined into one or more lower level goals, referring to the same target.

Arbitrary Refinement of Objectives: in this form of refinement of objectives, the goal and target are quite different from the higher-level objectives.

Procedures: where a policy may be refined by an unordered set of lower ones

Delegation of Responsibility: in this type of relationship, one subject delegates responsibility for the objective to another subject.

Related work in the area of policy hierarchies has been presented by [Wies94], specifying a four level hierarchy: Corporate or high level policies, Task-oriented policies, Functional Policies and Low-level Policies.

A declarative, object-oriented language has also been developed for specifying policies for the management of distributed systems, including constructs for specifying the basic types of policies described before [Ponder].

2.2 IETF Policy-based Networking

Two working groups in the IETF have considered policy management or *policy-based networking*: the Resource Allocation Protocol (RAP) Working Group (WG) and the Policy Framework WG.[1] The purpose of the RAP WG is to establish a scalable policy control model for RSVP and specify a protocol for use among RSVP-capable network nodes and policy servers. The Policy WG has provided several drafts describing a general framework for representing, managing, sharing and reusing policies in a vendor independent, interoperable and scalable manner as well as defining an extensible information model for representing policies and an extension to this model to address the need for QoS management.

The RAP WG has described a framework for policy-based admission control specifying the two main architectural elements [Yav00]: the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). PEP represents the component that always runs on the policy-aware node and it is the point where the policy decisions are actually enforced. The PDP is the point where the policy decisions are made. When a PEP receives a notification or a message that requires a policy decision, it creates a request that includes information which describes the admission control request. Then, the PEP may consult a local configuration database to identify which policy elements can be evaluated locally, passes the request with this set to the Local Policy Decision Point (LPDP) and receives the result. The PEP then passes all the policy elements and the partial result to the PDP which combines its result with the partial result from the LPDP and returns the final policy decision to the PEP.

[1]See: www.ietf.org/html.charters/rap-charter.html and http://www.ietf.org/html.charters/policy-charter.html

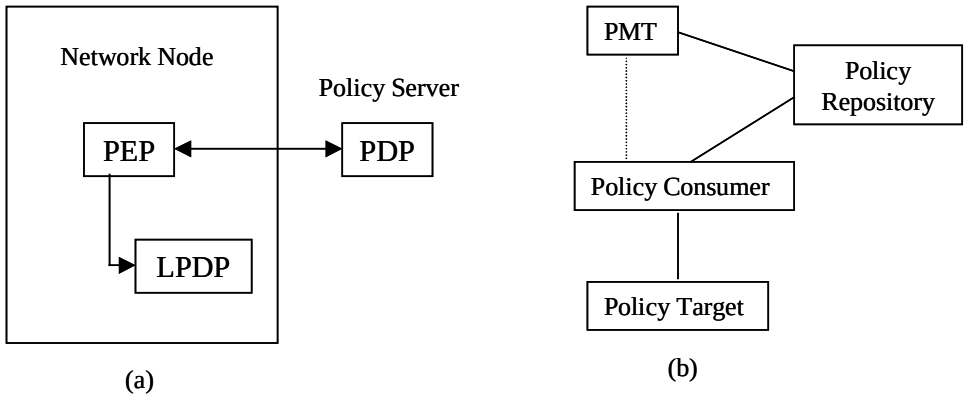

Figure 2 a) RAP WG Policy Framework for Admission Control, b) Policy WG Framework.

The Policy Framework WG defines policy as an aggregation of *Policy Rules*. Each policy rule comprises a set of conditions and a corresponding set of actions that are intended to be device- and vendor-independent. Policy Rules are of the form: if <condition> then < action>. The <condition> expression may be a compound expression and it may be related to entities such as hosts, applications, protocols, users, etc. The <action> expression may be a set of actions that specify services to grant or deny or other parameters to be input to the provision of one or more services.

The four major functional elements of the Policy Framework described by this group are:

- A **Policy Management Tool** to enable an entity to define, update and optionally monitor the deployment of Policy Rules.
- A **Policy Repository** to store and retrieve Policy Rules.
- A **Policy Consumer** which is a convenient grouping of functions, responsible for acquiring, deploying and optionally translating Policy Rules into a form useable for Policy Targets.
- A **Policy Target** which is an element whose behaviour is dictated by Policy Rules carrying out the action indicated by the Policy Rule.

A detailed description of the functionality of each element can be found in [Stev99].

3. Considerations on Policy-based Network Management

3.1 Policies as Means for Programmable, Extensible Management Systems

One of the key motivations behind policy-based management is flexibility and graceful evolution of the management system so that it can adapt to changing requirements over a long period of time. This is achieved by disabling / modifying old policies and by introducing new ones in order to meet changing requirements. A key aspect of a policy-based management system is that changes to *targets* should be performed in a consistent fashion, avoiding policy conflicts that may leave the managed system in an inconsistent state. Conflicting actions do not occur only in policy-based management systems but are potentially possible in any control system which performs intrusive management actions by *modifying* targets rather than simply *observing* them. Below we consider aspects of policies, intrusive management and conflicts in different management frameworks.

Enterprise networks are typically managed with SNMP, using a relatively simple management architecture consisting of a single, centralised “network management centre (NMC)”. The latter supervises network elements located typically in a cluster of local / metropolitan area networks. In this architecture, the elements are typically configured one-by-one, in an isolated fashion, through the supervision of a human network manager and according to an overall network operation policy, which is worked out beforehand. This means that (re-)configuration is infrequent and takes place manually.

In an evolution of this scheme, configuration parameters for every device are stored in a repository e.g. a directory, which is contacted by the devices upon cold or warm starts so that a device picks up necessary parameters and configures itself; this makes the system more scalable.

The NMC supervises, i.e. monitors, the managed devices, provides a view of the current network state, alerts the human manager in case of abnormal changes but does not attempt to reconfigure the network using automated logic. Reconfiguration is typically left to human managers who may modify first the network operation policy using their intelligence in order to overcome the problem. There are no conflicts in this architecture, they might only occur because of a wrong human-derived configuration policy but, with the right precautions, this should not happen. While this simple, centralised architecture with emphasis in monitoring rather than control, works adequately for best-effort IP networks, it cannot meet the needs of emerging multiservice networks with QoS guarantees. The latter require frequent, automated configuration changes according to a network-wide view, as presented in section 4.

Telecommunication networks are managed according to the hierarchically distributed Telecommunications Management Network (TMN) model. Initial and subsequent (re-)configuration of network elements occurs through element managers, which are orchestrated by a logically centralised but physically distributed, network manager. The latter has a view of the network-wide policy and implements it through automated logic by supervising the network elements and reconfiguring them in order to introduce new services or to recover from performance, fault and other problems. This management logic can be altered to a limited extent by modifying managed objects that model its operation. An example of TMN-influenced proactive and reactive management systems for ATM management can be found in [Georg99].

All configuration changes occur through a configuration manager, which holds the physical and logical network topology and partitioning. Requests coming from service, performance, fault and other managers are carefully validated, in order to maintain network consistence and integrity. Despite this validation, it is possible that different managers have conflicting configuration requirements. This can lead to inconsistent network state which satisfies only one of them, or to race conditions, in which the managers keep requesting changes to their preferred configuration state when they sense it has been changed back. Such conflicts can be avoided by careful modelling, design and testing of the management system, but conflicts may still occur at run-time when the system is stressed by real-world conditions not previously anticipated. This is rare though and also points to system integrity issues which are outside the scope of the paper.

Policy-based may be applied to both enterprise and telecommunication networks. The view taken by IETF seems to be compatible with the centralised model used in managing enterprise networks, though policy work in the research community has previously pointed to distributed models. In this discussion we will consider the centralised model to demonstrate the points and we will examine policies in distributed hierarchical systems in the next section.

The key aspect of a policy-based system is that management logic is expressed through declarative policies, evaluated in policy consumers. In the IETF model, the policy consumer can be thought as a centralised manager, with the execution of provisioning policies resulting in configuration operations on managed objects within network elements. In [Slom99], the policy consumer is seen as a hybrid manager-agent where the policies express the manager intelligence and access the co-located managed objects of the agent part; we consider and extend this model in the next section. The essence though is that management intelligence can be modified, added and removed by manipulating policies as requirements change. In policy-based systems, management intelligence does not follow the rigid analysis, design, implementation, testing and deployment cycle, and as such, conflicts may be the norm rather than the exception. Conflict analysis and detection is required both statically, at policy introduction and deployment time, and also dynamically, at run time. Policies are often associated with interpreted logic but we believe their salient characteristic to be the composition of a system

from building blocks which can be introduced, modified and withdrawn at any time, without having rigorously tested the resulting system in every such modification.

Taking the policy approach to the extreme, all management intelligence could be policy-based, starting with a system which comprises only manageable network elements and policy consumer capabilities in the role of an "empty" centralised manager. This is the complete opposite of the rigid, hierarchical TMN approach, but results in a pretty undefined and fluid system, which will be very difficult to protect against conflicts, or to even realise it with declarative policies in the first place. We see policies mostly as a means to "late bind" functionality to an existing management system, which is hierarchically distributed in order to meet the management needs of multiservice networks. In this case, policies can be seen as a means to achieve "programmability" of the system with new functionality and lead to a flexible system that can cope with evolving requirements. We feel this is a much more realistic proposition than a purely policy-based approach for complex management systems. Until now it is not clear how policies can be used in the context of a hierarchical system. In the next section we consider policies that follow and mirror the hierarchical system decomposition.

3.2 Hierarchical Policies

In hierarchical management systems, hybrid agent-manager applications exist at different levels of the hierarchy, managing ultimately network elements at the lowest level. Manager-agent or managing-managed interactions occur top-down and possibly peer-to-peer but never bottom-up. A hierarchy may be *strict*, in which case the management layer N+1 builds on the functionality and services of layer N, or *relaxed*, in which case layers may be bypassed. In the following discussion we will assume a strict hierarchy for simplicity.

At layer N of the hierarchy, an agent-manager application comprises:

- Managed objects presenting the management capabilities of the application to the layer N+1 (or to the same layer N for peer-to-peer interactions).
- Management logic accessing managed objects of the layer N-1 (or of the same layer N for peer-to-peer interactions).

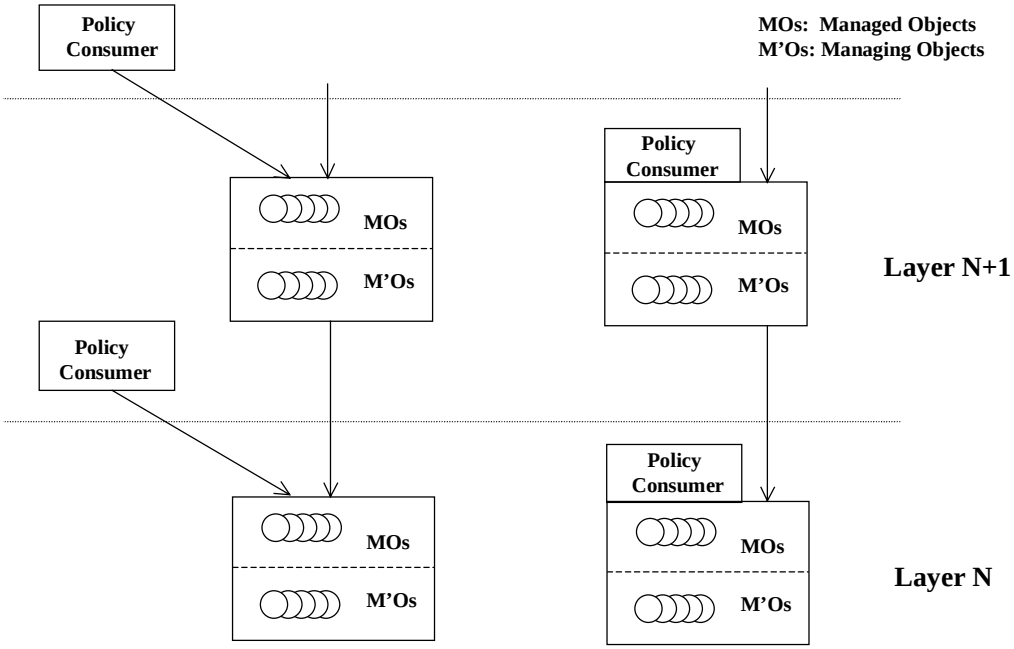

Figure 3 Hierarchical management with loosely (left) and tightly-coupled (right) policy consumers.

The managed objects constitute the top, i.e. agent, part of the application (see Figure 3), which is why we prefer the term agent-manager as opposed to manager-agent used in the literature. The managed objects and associated managing logic or managing objects represent "static" management intelligence, following a rigorous analysis, design, implementation, testing and deployment cycle. Parametrisation of the functionality of such an agent-manager application is possible to a limited extent by configuring managed object values. The deployment of such a hierarchy takes place bottom-up but the decomposition and design of the whole system takes place top-down, according to the management services to be provided.

The simplest form of introducing policies in such a system is through a separate policy consumer point where policies execute and access managed object at various different layers of the management hierarchy. In other words, the policies manipulate targets, i.e. managed objects, at all the layers of the hierarchy. The problem with this approach is that the policies are monolithic, logically and physically centralised, operating on a hierarchical distributed system. A better approach would be to structure the policies hierarchically, mirroring the system hierarchy, as explored next.

In a hierarchical policy system, policies at layer N+1 operate on managed objects of layer N. This implies in fact that these policies may be considered as part of the managing intelligence of layer N+1, in addition to the static intelligence of agent-manager applications. This approach is shown in the left of Figure 3. If these policies access managed objects in more than one layer N agent-managers, they could execute at a layer N+1 consumer point which complements the managing intelligence in this layer. If though they access managed objects in a single subordinate agent-manager, they could execute *at* that agent-manager, having local access to managed objects. In this case, the manager-agent at layer N is programmed with policy logic that belongs conceptually to the layer N+1 but since it relates to a particular agent-manager of the layer below, it has actually "migrated" there. This is shown in the right part of Figure 3 where the two policy consumers have migrated and now form an integral part of the agent-manager they relate to. In this paradigm, every agent-manager may potentially become a policy consumer, including of course ultimately the agents within network elements.

We view policies as complementing the static management system intelligence. One key aspect when designing a policy-capable hierarchical system is how much intelligence should be realised in a static fashion. Static intelligence should offer enough functionality to allow relatively easy extension of the system through policy logic but not too much so that there is still flexibility in terms of changing requirements. In principle, higher amount of static intelligence leads to a more rigid, less extensible but potentially more stable system while less amount of static intelligence leaves the system fluid, easily extensible but may result in instability as more and more functionality is realised through policies (more frequent conflicts etc.)

A key aspect in such a hierarchical system is policy refinement and this should naturally follow the hierarchical composition of the system. Policies may be introduced at any level but higher-level policies may possibly result in the introduction of related policies at lower levels. In a similar fashion to the bottom-up deployment of a static hierarchical system, policy hierarchies should be introduced in a bottom-up fashion, maintaining the completeness and integrity of the policy space. Policy refinement and transformation is a process analogous to software system analysis and design but in the context of a hierarchical system of a specific nature, e.g. IP Differentiated Services (DiffServ) network management, guidelines may be devised and followed.

We are thinking in particular of policy classification specific to a problem domain, going further than the general-purpose classification encountered in the literature. For specific classes of policies, we are thinking of policy refinement guidelines and rules that will assist and possibly automate refinement of policy instances. We are in fact envisaging situations in which changing parameters of a high level policy will result in changes throughout the policy hierarchy.

In the next section we present first the functional architecture of a distributed hierarchical system for IP DiffServ; this is designed from the beginning with policy extensibility in mind. In this system we

are thinking of network dimensioning, resource management and admission control policies. We then explore further the hierarchical nature of dimensioning and resource management aspects, presenting an elementary example of hierarchical policy decomposition.

4. Hierarchical Policy-Enabled IP Network Management

4.1 The TEQUILA Functional Architecture

The objective of the TEQUILA project (Traffic Engineering for QUality of Service in the Internet, at LArge Scale)[2] is to study, specify, implement and validate a set of service definition and traffic engineering tools in order to obtain quantitative end-to-end Quality of Service guarantees through careful dimensioning, admission control and dynamic resource management in Differentiated Services [Blake98] IP networks. The technical areas addressed by the project are: (a) the specification of static and dynamic, intra- and inter-domain Service Level Specifications (SLSs), (b) protocols and mechanisms for managing (negotiating, monitoring and enforcing) SLSs, and (c) intra- and inter-domain traffic engineering schemes to ensure that the network can cope with the contracted SLSs – both within domains, and in the Internet at large. The rest of this paper assumes a basic understanding of the DiffServ architectural framework and terminology.

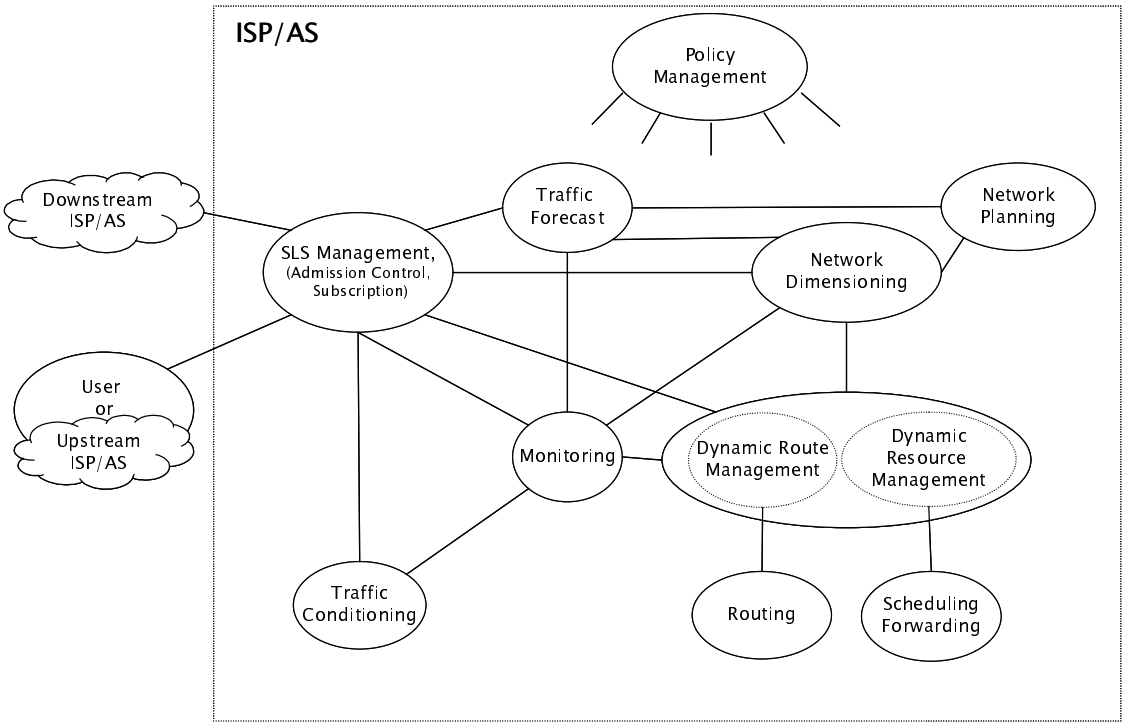

Figure 4 The TEQUILA functional architecture.

In order to achieve its technical goals the project has defined a functional architecture shown in Figure 4 [D1.1]. There are two main parts in this architecture, the SLS-related part and the Resource Management part. The first includes the SLS Management Functional Block (FB), which can be further decomposed to the Subscription, Admission Control and Interdomain SLS Request blocks. It also includes the SLS-related part of Monitoring and the Traffic Conditioning block. This part of the overall functional model is responsible for subscribing and negotiating long-term SLSs with users or other peer Autonomous Systems (ASs) and it performs admission control of dynamic SLSs. The

[2] http://www.ist-tequila.org

Traffic Conditioning FB classifies and marks packets according to negotiated SLS, as well as performs metering policing and shaping.

The other major part of the architecture concerns resource management. The Planning FB is responsible for long-term (order of months or years) planning of physical resources. Network Dimensioning works in order of days or weeks. This functional block is responsible for mapping the traffic onto the physical network resources and it configures the network in order to accommodate the forecasted traffic demands. The Network Dimensioning FB defines Multi-Protocol Label Switching (MPLS) paths at layer 2 and uses pure layer 3 capabilities in order to accommodate the expected traffic. The Traffic Forecast FB gets information from the current SLS subscriptions, traffic projections and historical data provided by the Monitoring FB, and uses traffic and economic models in order to provide the appropriate forecasted traffic matrices to the Dimensioning and Planning FBs. Based on constraints/rules provided by Network Dimensioning, the Dynamic Route Management FB modifies (in a timescale of minutes or hours) routing parameters in routers. If MPLS is used it dynamically adds/merges/splits/reroutes paths while there is an equivalent behaviour in the case of pure layer 3 routing. This functional block adjusts routing parameters (weights, load distribution on equal multi-paths) or modifies MPLS paths. The Dynamic Resource Management FB sets buffer and scheduling parameters on links according to network dimensioning directives and constraints. It also allocates capacity to existing/newly created paths. The Routing FB is Constraint-based, DiffServ class-aware and uses constraints to reduce algorithm complexity and hence reduce the convergence time. The Scheduling and Forwarding FB implements Per-Hop-Behaviours (PHBs), e.g. Expedited/Assured/Default Forwarding (EF/AF/DE), using buffer management and scheduling mechanisms. The Monitoring FB performs monitoring and measurements in various levels of abstraction. This FB has both network-wide and detailed per node view of the network, and operates as an agent to other FBs of the architecture, providing the necessary monitoring information they request.

The Policy Management FB is an essential part of the TEQUILA system and is described in more detail in the following section.

4.2 Hierarchical Policy Management in the Tequila System

The policy functional block in the functional architecture includes (see Figure 5) the Policy Management Tool, the Policy Storing Service and the Policy Consumers which correspond to their associated functional blocks (note “A Tequila Functional Block” in Figure 5), e.g. SLS related admission policies for the SLS Management block, dimensioning policies for the Dimensioning block, dynamic resource/route management policies for the Dynamic Resource Management block, etc.

In this model there exist many Policy Consumers, associated with particular functional blocks of the hierarchical management structure. Targets can be the managed objects of the associated functional block or of lower-level functional blocks (but never of higher-level blocks). Policy Consumers need also to have direct communication with the Monitoring functional block in order to get information about traffic-based policy-triggering events. Note that triggering events may be also other than traffic-related, in which case it is typically generated by the specific functional block with which the Policy Consumer is associated.

Policies are defined in the Policy Management Tool, which provides a “policy creation environment”. Policies are defined in a high-level language, are translated to object-oriented policy representation (information objects) and stored in the policy repository (Policy Storing Service). New policies are checked for conflicts with existing policies, although some conflicts may only be detected during execution time. After the policies are stored, activation information may be passed to the associated Policy Consumer.

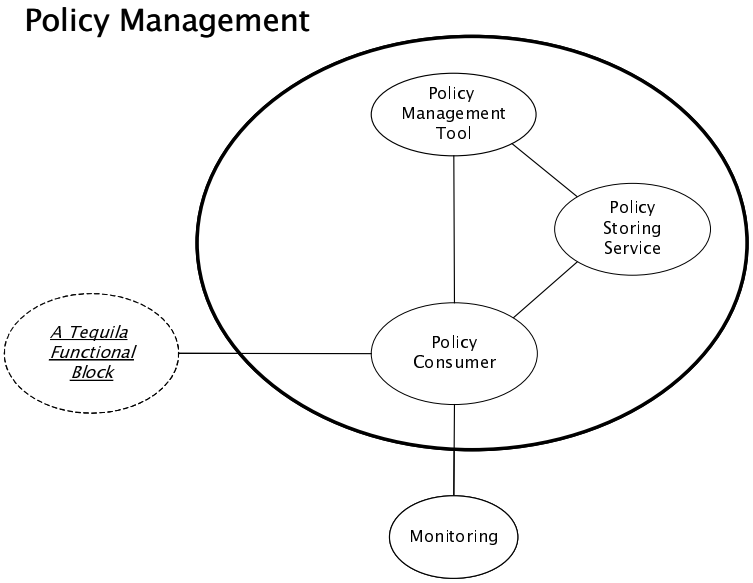

Figure 5 Decomposition of the Policy Management Functional Block.

Every time the operator enters a high level policy, this should be refined into policies for each layer of the TEQUILA functional architecture forming a policy hierarchy that reflects the management hierarchy. As mentioned in the literature [Moff93], it is very difficult to support an automated decomposition of a policy without human intervention. The administrator should define generic classes of policies and provide some refinement logic/rules for the policy classes that will help the automated decomposition of instances of these classes into policies for each level of the hierarchical management system shown in Figure 4. These generated policies can be interpreted and enforced by the Policy Consumer associated with the responsible functional block (agent-manager). The Policy Consumers will retrieve the policies from the Policy Storing Service in a format so that all the necessary information can be mapped to the specific set of functions of the FB (agent-manager). This is done by setting the appropriate parameters to influence or modify the agent-manager's behaviour and by registering the appropriate events in order to be notified about when this Policy must be enforced. This refinement logic can be stored in the Policy Management Tool and all the policies of the hierarchy will be generated at the tool, stored in the repository and then retrieved by the responsible policy consumers. Another possibility could be to distribute the refinement rules in the appropriate consumers so that each one will be responsible for generating and sending the next lower-level policies to the consumers that reside in the next lower layer of the management system.

4.3 An Example of Hierarchical Policy Decomposition

Let's assume that the administrator of an AS wants to ensure availability of a specific traffic class for a time period, for example s/he wants to enforce the following policy[3]:

"At least 10% of Network Resources should always be available for EF traffic" (1)

In order for this policy to be enforced, it must be refined into policies that apply to each layer of the management architecture. Before specifying this policy the operator must go through the following three steps so that automated refinement can be supported:

1. Define the template of the generic policy class.
2. Define the range of parameters/attributes of the policy class.

[3] We do not assume any specific Policy Definition Language in this example.

3. Provide the guidelines and rules for the refinement.

Policy (1) is an instance of a generic policy class. This resource management policy class has four parameters and has the following generic form:

<bound> <percentage> of Network Resources <period> available for <traffic type> (2)

As described above, the human operator should first define the template of the generic class (2) and then the range of the parameters/attributes that can be specified for a valid instance of this policy class (see Table 1). In order to create an instance of this class, such as (1), the operator has to edit the above template and set the required parameters to specific values (e.g. bound = "≥", percentage = "10%", period = "from 9am to 2pm", traffic type = "EF"). If the parameters specified are in the previously defined range of values then the procedure for enforcing this policy will normally proceed, otherwise the validation function of the Policy Management Tool will return an error. Table 1 shows the valid range of values of the parameters in the above resource management policy class:

Table 1 Range of values for policy parameters of the resource management policy class.

Parameter – Attribute	**Range of Values**
Bound	≤,≥
Percentage	0-100
Period	Any time period expression
Traffic type	EF, AFxy, BE, CS1-8

The final step is to provide some refinement rules and logic that will assist the automated decomposition of *every* instance of the resource management policy class (2) to policies for each level of the hierarchical management system as shown in Figure 6.

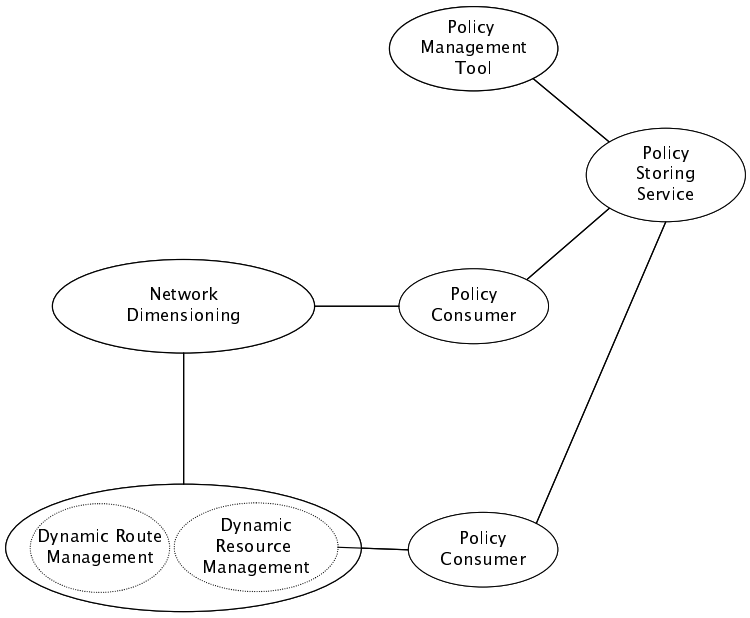

Figure 6 Enforcement of the Resource management policy at each level of the Tequila Functional Architecture.

The functional blocks that are responsible for enforcing any instance of policy class (2), are Dimensioning and Dynamic Resource Management. Assuming a strict management hierarchy in the Tequila functional architecture, any function of the Dimensioning functional block will operate on managed objects of the Dynamic Resource Management functional block. Consequently, high-level resource management policies will be decomposed into dimensioning and dynamic resource management policies. In this architecture, the policy consumers will be eventually tightly coupled with their respective functional blocks (or agent-managers), in a similar fashion to the model depicted in the right part of Figure 3.

More specifically, in the Dimensioning functional block, every instance of the resource management policy class (2) is enforced as follows: when the generated policy for this layer of the management architecture is downloaded to the Policy Consumer, at the time <period> when this policy is active, it must first check the current configuration, i.e. how the network is already dimensioned. This is done by passing the appropriate parameters taken from the policy instance regarding the <bound>, <percentage> and <traffic type> to a `check_current_configuration()` method that will trigger a re-dimensioning of the network if it is needed. Then, when re-dimensioning is required, i.e. when network/traffic conditions are such that Dynamic Route and Resource Management algorithms are no longer able to operate effectively, at the time <period> the policy instance is enforced by passing the parameters to the `check_forecast()`method that will check the traffic matrices retrieved from the traffic forecast functional block, and are input to the dimensioning algorithm. If the <percentage> allocated to the traffic type specified in the policy instance does not violate the policy, then there is no need to influence the dimensioning algorithm. Otherwise, if the traffic matrices are not compatible with the policy then the dimensioning algorithm function must be executed with a different/modified traffic matrix. For the specific policy instance (1), the `check_current_configuration()`method will trigger a re-dimensioning only if the current configuration is such that less than 10% of resources are allocated to the EF traffic class. Moreover, at re-dimensioning, the `check_forecast()`method will modify the traffic matrices so that at least 10% of resources are allocated to EF.

After the network is configured according to the dimensioning output, Dynamic Resource Management is responsible for adjusting the resources according to the dynamic changes in traffic demand. The dynamic adjustment of resources might result into situations where the policy is no longer enforced; therefore we need to make this functional block policy aware.

The generated policy for this layer of the functional management architecture is enforced in the Dynamic Resource Management block as follows: at the time <period> that the policy is active, the Policy Consumer should pass the needed parameters (<bound>, <traffic type>) to the `check_set_operation()` method and this will indicate whether the operation performed by the Dynamic Resource Management block is allowed or not. Actually, this will check through the `set_resources()`method if the addition or removal of resources allocated to the <traffic type> violates the specified policy. For example, assuming the policy instance (1) of the resource management class (2), additional resources allocated to the EF traffic will always be allowed and deletion of resources will only be performed if the overall allocation of resources for EF is greater than 10%.

In the previous example we have assumed that the Dynamic Resource Management Block is logically and physically centralised. If the functionality of the block is physically distributed to reflect the distribution of the network being managed, agent-managers will contain replicas of functions. This makes the enforcement of the above policy instance more complex since an overall knowledge of the modifications of the allocated resources to the traffic type is needed. This can be done either by adding some cooperation ability to the distributed agent managers or by enforcing the policy in a non-optimal way, restricting each agent-manager to cause a less than 10% allocation of resources to EF traffic in the local managed area of the network.

From the above example one may observe that if the functional architecture was not designed to be policy aware, some methods of the functional blocks such as `check_current_configuration()`and `check_forecast()` in Dimensioning and `check_set_operation()`in Dynamic Resource Management would be redundant. Though obvious, it should be mentioned that these methods are part of the computational interface of the respective block or may be derived from other methods, which are part of that interface. It is for further investigation whether this logic could be automatically generated and downloaded to the appropriate blocks or it should be hardwired beforehand in the agent-managers. The whole area of hierarchical "policy-aware" system specification/design and the subsequent population of that system through hierarchical policies with (some) automated support in policy-refinement constitute the core of our future research in this area.

5. Summary and Future Work

While most research work on policies has concentrated in necessary fundamental aspects such as classification, policy language, conflict detection and most recently in policy representation and storage, little work has been done in the area of the harmonic coexistence of policies and traditional, possibly hierarchical, management systems. Policies, apart from their high-level declarative nature, can be also seen as a vehicle for "late binding" functionality to management systems, allowing for their graceful evolution as requirements change. It is this aspect of policies we find most interesting and we have been exploring the potentiality of designing "policy-aware" management systems, in which a line has to be carefully drawn between "hard-wired" functionality and policy logic.

In this paper we first described the salient characteristics of policy-based management and we explored their coexistence with hierarchical management systems, presenting first an initial version of a generic framework and showing then how a system for IP Differentiated Services management can be designed and built using such a framework. There are many issues that are still unresolved but the fundamental target is to be able to come up with a system that will be able to sustain requirement changes and evolve gracefully through policies without any changes to its carefully thought-out, "hard-wired" initial logic. We are interested in deriving generic guidelines on how this can be done and also guidelines on hierarchical policy decomposition and refinement. We are not sure if such guidelines can be problem domain independent but we hope at least to produce such guidelines in the context of IP Differentiated Services management.

As a continuation of the work presented in this paper, we will be concentrating in the definition of an object-oriented information model representing the capabilities of each layer of the hierarchical functional architecture that was described in section 4.1. This information model will assist us in the definition of QoS policies. Realisation of this model will be done by parametrising the management functions of the hierarchical management system, following, initially in a bottom-up approach to capture each layer's functionality and then in a top-down analysis to record the policy-management requirements. Moreover, we will be focusing on the specification of dimensioning, resource and route management policies, by defining generic policy classes for specific resource management cases. We will further explore the concept of the automated decomposition and transformation of instances of these policy classes by providing specific guidelines. Finally a realisation of the above concepts will be demonstrated by using a prototype implementation. We will report our findings in the future.

Acknowledgments

Part of this work was undertaken in the context of the IST TEQUILA project, which is partially funded by the Commission of the European Union. The authors would also like to thank David Griffin of University College London for his participation in the initial discussions towards the formulation of some of the ideas mentioned in this paper.

References

[Blake98] S. Blake et al, *An Architecture for Differentiated Services*, Informational RFC-2475 December 1998.

[D1.1] D. Goderis et al., *Functional Architecture and Top Level Design*, TEQUILA Deliverable D1.1, September 2000.

[Georg99] P. Georgatsos, D. Makris, D. Griffin, G. Pavlou, S. Sartzetakis, Y. T'Joens, D. Ranc, *Technology Interoperation in ATM Networks: the REFORM System*, IEEE Communications, Vol. 37, No. 5, pp. 112-118, IEEE, May 1999.

[ICpol] Imperial College, *Policies for Network and Distributed Systems Management*, http://www-dse.doc.ic.ac.uk/policies/.

[Moff93] J. Moffett, M. Sloman, *Policy Hierarchies for Distributed Systems Management*, IEEE Journal on Selected Areas in Communications, Vol. 11, No. 9, pp. 1404-141, December 1993.

[Ponder] N. Damianou, N. Dulay, E. Lupu, M Sloman, *Ponder:A Language for Specifying Security and Management Policies*, Imperial College Research Report DoC 2001, January 2000.

[Slom89] M. Sloman, J. Moffet, *Domain Management for Distributed Systems*, Integrated Network Management I, B. Meandzija and J. Westcott, eds., pp. 505-516, North Holland, 1989.

[Slom94] M. Sloman, *Policy Driven Management For Distributed Systems*, Journal of Network and Systems Management, Vol. 2, No. 4, pp. 333-360, Plenum Publishing, December 1994.

[Slom99] M. Sloman, E. Lupu, *Policy Specification for Programmable Networks*, Proc. of the 1st International Conference on Active Networks, Berlin, Germany, ed. S. Covaci, Springer Verlag, June 1999.

[Stev99] M. Stevens et al., *Policy Framework*, Internet Draft, draft-ietf-policy-framework-00.txt, September 1999.

[Wies94] R. Wies, *Policy Definition and Classification: Aspects, Criteria, Examples*, Proc. of the 5th IEEE Workshop on Distributed Systems Operations and Management (DSOM), Toulouse, France, October 1994.

[Yav00] R. Yavatkar, D. Pendarakis, R. Guerin, *A Framework for Policy Based Admission Control*, Informational RFC 2753, January 2000.