Considerations on Policy-based Network Management

P. Flegkas, P. Trimintzios, I.Andrikopoulos, G. Pavlou
Centre for Communication Systems Research, University of Surrey
{P.Flegkas, P.Trimintzios, I.Andrikopoulos, G.Pavlou}@eim.surrey.ac.uk

Abstract: *Policy-based Management has been the subject of extensive research over the last decade. 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. In this paper we are 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.*

1. Introduction.

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.

2. Policies as Means for Programmable, Extensible Management Systems.

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.

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 [1].

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 [2]. In [3], 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. 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).

The managed objects constitute the top, i.e. agent, part of the application (see Figure 1), 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.

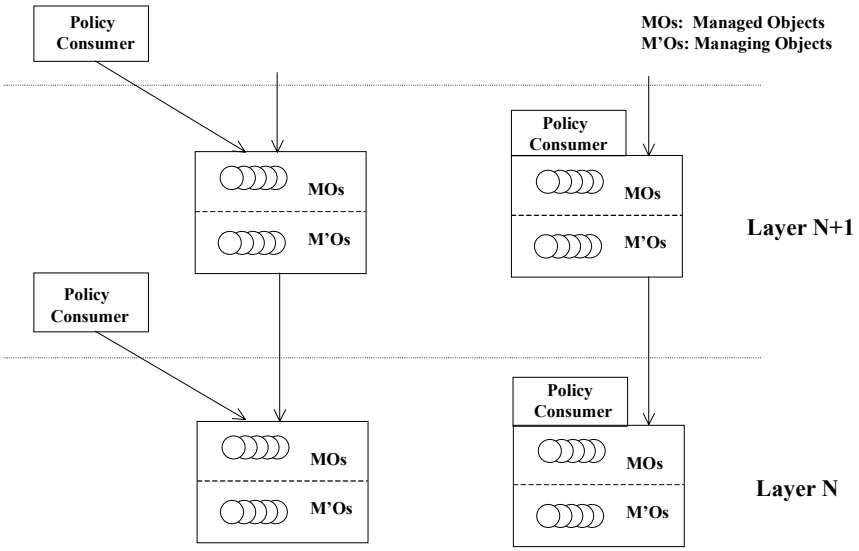

Figure 1: Hierarchical management with loosely (left) and tightly-coupled (right) policy consumers.

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 1. 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 1 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.

4. Summary.

In this paper we first described the salient characteristics of policy-based management and we explored their coexistence with hierarchical management systems, presenting an initial version of a generic 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.

References

[1] 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.

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

[3] 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.