

Towards implementing policy-based systems management

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

1996 Distrib. Syst. Engng. 3 78

(<http://iopscience.iop.org/0967-1846/3/2/002>)

View [the table of contents for this issue](#), or go to the [journal homepage](#) for more

Download details:

IP Address: 38.107.179.210

The article was downloaded on 20/02/2012 at 07:45

Please note that [terms and conditions apply](#).

Towards implementing policy-based systems management

Bernd Meyer[†], Frank Anstötz and Claudia Popien

Department of Computer Science (Communication Systems), Aachen University of Technology, Ahornstr. 55, 52056 Aachen, Germany

Abstract. Managing modern information systems becomes increasingly complex. Therefore, the need for flexible mechanisms which assist system managers is evident. Management policies are means to control object behaviour. Considerable work has been done on the specification and classification of policies, but their enforcement is still an open research topic. In this paper we propose an event-driven approach to policy-based management, where a manager delegates policy enforcement to an agent. We introduce a policy controller object, which holds management information in the form of rules and controls the behaviour of arbitrary objects. We also present performance results and an example using the introduced concepts which is implemented on the ANSAware distributed platform.

1. Introduction

As modern information processing systems evolve from large mainframe computers to large networks of interconnected workstations, designing, implementing and maintaining application and system software becomes even more complex and difficult. While current research is focused on design and implementation of distributed platforms and applications, this work is concerned with managing distributed systems. Especially the dynamic nature of distributed systems forces management to provide maximum flexibility. Currently most work on management systems deals with network components, e.g. OSI Management or Telecommunications Management Networks, see [1] or [2]. Whereas OSI Management Systems for network components assume a central administration unit, distributed systems management has to be realized in a distributed fashion in order to obtain sufficient scalability.

Usually management decisions are made by human system administrators. In order to achieve a quicker response to system changes, it would be desirable to delegate some decisions to the management system. Therefore, a system manager should provide the management system with some information on how to make decisions, i.e. a 'policy'. A policy relates permitted, prohibited or required behaviour to objects. Policies can be defined on different levels of abstraction thus forming a policy hierarchy. While abstract policies deal with enterprise goals and business activities, lower-level policies are bound to the interfaces' functionality of a concrete computing system. In general, it is not sensible to relate a policy to each individual object, but rather to assign a common policy to a group of objects. Therefore, the notion of a domain has been introduced. Domains also contribute to the scalability, since they allow management applications to deal with a smaller number of

objects within a large distributed system. Currently distributed systems management based on domains and policies is an important research issue. A lot of work has been done on domain and policy services within two ESPRIT research projects called IDSM and SysMan, see [3].

Our approach concentrates on mechanisms to enforce policies by means of event-triggered rules. After outlining policy-based management in section 2, we introduce a new approach to policy enforcement in section 3. Policies are implemented by the Policy Control Object (PCO), which is linked with the object under control via service and management interfaces. It interprets rules that are compiled from policy definitions. We also present measurement results on the performance delay caused by the PCO. In section 4 we employ our policy enforcement approach to the management of a multi-interface server.

2. Basics of policy-based management

A distributed system consists of a set of cooperating objects. Each object is able to interact with other objects by invoking operations at the other object's interface. In general, objects can have more than one interface. The interface encapsulates the internal object state that can only be observed and manipulated via interface operations. The above given system model outline is based on concepts of the Reference Model of Open Distributed Processing [4]. In the following we assume that each object offers a service and a management interface. Interactions between managers and agents are either notifications sent to a manager or operations invoked with the agent.

2.1. Policies

If we take into account that there may be a large number of objects in the system, the need for manager support in

[†] E-Mail address: bernd@i4.informatik.rwth-aachen.de

controlling object behaviour becomes evident. One possible approach is the use of management policies. Policies provide a means to define goals of an organization and describe how to achieve them. When we consider a distributed system which consists of a collection of objects, applying a policy means controlling object behaviour. In order to use policies, it is necessary to find a suitable definition and representation form.

First we outline the policy structure identified in [5]. A policy obliges a subject, i.e. a manager, to enforce a certain goal on an object. Subject and object can either be assigned to certain objects or to domains. A key policy concept is the modality. Modalities divide policies into authorization and motivation policies. Whereas authorization policies are concerned with goals that are permitted or prohibited, motivation policies define goals to be enforced or to be deferred.

As mentioned above, policies can be defined on different levels of abstraction, see [6] and [7]. We define high-level policies to describe goals to be achieved by a certain object or management application. High-level policies only describe what is to be achieved and not how this is going to happen. Therefore, medium-level policies describe strategies to achieve high-level goals. Although medium-level policies are less abstract than high-level ones, they are not bound to a certain computing or management system. These platform dependencies will be introduced by refining medium-level policies into low-level policies. From this point of view, medium-level policies describe abstract behaviour, whereas low-level policies are concrete representations of the same policy.

Another important aspect is the persistence of the policies. This implies that policy behaviour is different from programming code. Which actions are required for a given policy to be satisfied is one of the main problems arising in policy management. As policies can be expressed very generally, it is evident that the satisfaction problem is difficult to solve.

2.2. Example for policy refinement

The policy P1 gives an example. We consider a distributed system which consists of a server process and several users with client processes sending jobs to the server.

P1: The manager must ensure that a server offers reasonable response time to users.

Before we can apply this policy, further analysis is required, for example we need to know how ‘reasonable response time’ is interpreted and how it can be achieved. Thus, the analysis of policies may require a considerable amount of knowhow which complicates the automation of policy management. One approach to address this problem is the introduction of policy hierarchies like in [6]. A policy hierarchy is formed by a high-level policy which is then refined in several steps into lower-level policies. On the lowest level, the resulting policies are called procedures. A procedure represents an ordered or unordered sequence of actions which can be carried out without further analysis. For example, policy P1 can be refined as follows:

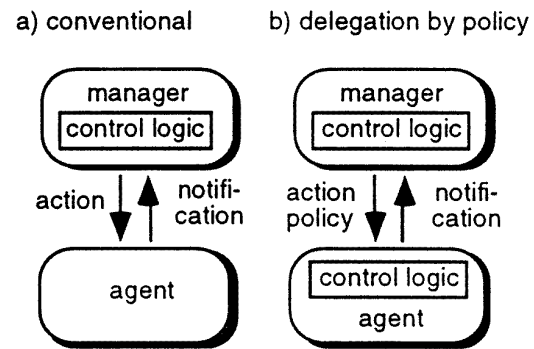


Figure 1. Different manager-agent models.

P2: The service time for each job must be minimised and load must be reported daily.

Further refinement leads to policy P3:

P3: The server has a queue with maximum length of 20. If the queue is filled with jobs the next incoming job will be refused. When the current load is larger than 0.8, the server strategy must be changed from FIFO to SEPT. If the load drops below 0.75, the strategy is changed back to FIFO. A load report must be sent to the manager every day at 09:00 a.m.

Policy P3 is on a level which allows the satisfaction of policy goals without further refinement. In P3 SEPT means ‘shortest expected processing time first’. In the following section we present a component for policy-guided control of object behaviour.

3. Enforcing policies by event-triggered rules

This section describes the structure and components of the policy enforcement system we are currently developing at Aachen University of Technology. Current network management systems assume that all control logic resides within a single manager (see figure 1(a)).

The task of the agent is collect management-relevant information, condense and filter it, and forward it to managers. In contrast to managers they cannot make decisions and invoke operations on objects. We propose an alternative approach that we call *delegation by policy*, where the manager delegates parts of its control to the agent by means of a policy, see figure 1(b). The agent itself is able interpret the manager’s policy.

3.1. Formalizing policies

First of all, a policy has to be described formally. Therefore, we use a special language—the Policy Definition Language (PDN) as defined in [8] and [9]. It is a general notation for policies used by management systems and enables flexible behaviour specification. For example, the

goals are represented by a behaviour description of the corresponding objects, and the policy modality determines whether this behaviour is forced, permitted or prohibited.

From this behaviour description, we have to generate a collection of semantically equivalent rules. This means, if these rules are applied to an object, it will behave according to the specified behaviour (and therefore achieve the policy goals). Assuming that the policy is on a procedural level, this implies that

- policy subjects and target objects denote either entities which are present as system objects or are given implicitly,
- the behaviour which represents the policy goals can be achieved directly by sequences of actions without further interpretation and refinement,
- the policy modality is evident (in PDN, the modality is expressed by keywords like *force*, *permit*, *prohibit* and *restrict*),
- the constraints can be expressed as conditions on local management information.

Furthermore, we must know the structure of the management interface of the objects considered. In detail, we need information about

- interactions occurring at the object's service interface including their parameters,
- operations allowed to be invoked on management attributes
- management operations directly related to attributes, and
- notifications the object may send to the manager.

3.2. The rule concept

We have adapted the trigger concept from the area of active databases, where it is used for automatically controlling integrity constraints, see e.g. [10] or [11]. Therefore, so-called event condition action (ECA) rules are used. It should be noted that the area of rule based control systems is not yet mature. Several issues like termination, confluence or observable determinism in rule systems are not yet solved, see [12] and [13].

In the following we distinguish three event types, message, threshold and timer events. Message events are triggered when interactions occur at the service interface of an object. Each message event contains an identifier for the calling or called object, the service interface identifier and the interaction name with its parameter values. The threshold event is triggered if the management attribute either reaches, falls below or exceeds a given value. Last but not least, timer events are used to model time dependencies like alarm times or time-dependent constraints. Timer events have a certain precision, e.g. one minute. This precision depends on the reaction time of the management system, i.e. if it takes 10 ms to select and fire all matching rules, it makes no sense to have timer events every millisecond. Often it is useful to fire rules only when a certain condition is satisfied. Conditions contain constraints on the value of management attributes

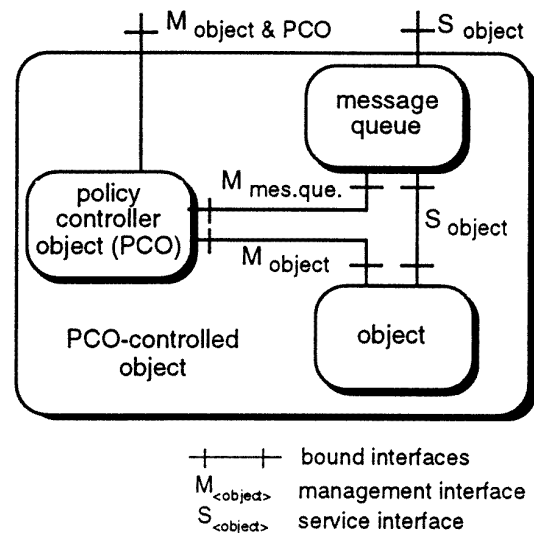


Figure 2. Objects related to a Policy Controller Object.

or parameters passed by interactions. The action part of a rule is described by an arbitrary set of interactions between the PCO and the management interface of managed objects. As a consequence of an action, new events can occur.

3.3. The Policy Controller Object

For adding control logic to objects, we extend each object with a Policy Controller Object (PCO), see figure 2. The PCO controls the object behaviour according to a given policy received from a manager. In addition, the object needs to be equipped with a management interface and a separate queue for service interactions, e.g. incoming operation invocations. This message queue emits notifications to the PCO, e.g. when a new job has arrived, and allows a manager to either refuse or accept a certain job in the queue. In the positive case it will forward the invocation to the object's service interface. The PCO consists of three parts, see also figure 3:

- the **interpreter**, which receives event notifications and policies (as rules) from managers and is able to invoke operations at the object's management interface,
- the **rule table**, which stores all relevant motivation policy rules,
- the **access control information**, that stores authorization policies like e.g. access control lists.

A policy is idle as long as there is not policy assigned to it. In this case the object behaviour is not affected at all. To enable policy-based management, a manager must formulate the desired policy, translate it into rules and delegate these rules to the PCO of the object to be managed. The PCO receives the rules through its management interface, see interface $M_{\text{object}\&\text{PCO}}$ in figure 3. At this interface, operations are defined which enable the client to set, delete, suspend and activate a policy.

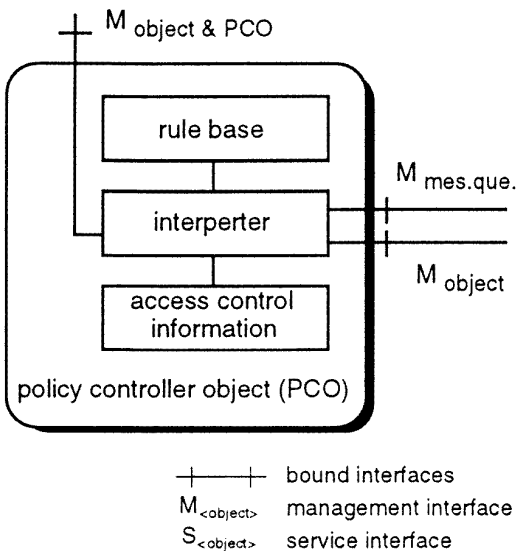


Figure 3. Structure of a Policy Controller Object.

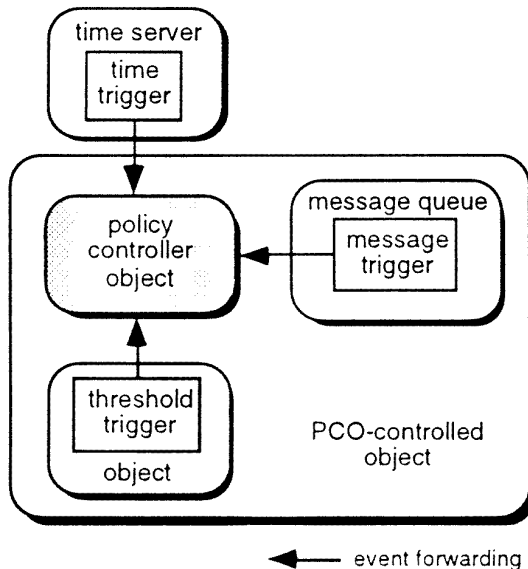


Figure 4. Event-forwarding triggers.

3.4. Implementing event-forwarding triggers

Three different triggers are used to detect the three event types and generate notifications, see figure 4. The message trigger is activated when an interaction occurs at the object's service interface. Threshold triggers are related to the object's management attributes. Changing objects must also modify the corresponding attribute values. When an attribute value is changed, the threshold trigger checks the current attribute value. If a given value is reached, a notification is generated. The time trigger keeps track of alarm times, therefore it needs to be supported by a time server. When the time server indicates that a certain alarm time is reached, a notification is sent to the PCO.

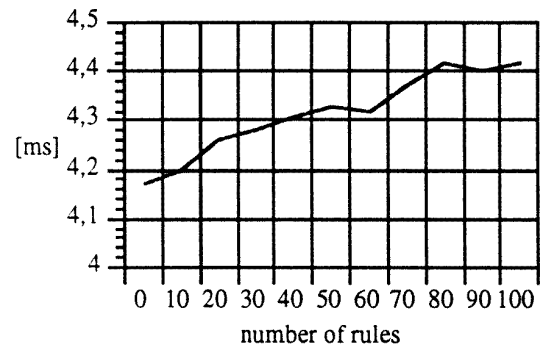


Figure 5. Average delay in milliseconds caused by rule inspection.

3.5. The interpreter

The interpreter receives notifications from triggers and activates the corresponding rules in order to enforce the desired policy behaviour. Rules are stored in separate tables depending on their event type. A policy identifier is added to each rule for the purpose of policy handling, because in general a policy is translated into a set of rules. A notification is passed to the PCO locally (in the case of threshold event or message event the corresponding triggers are located locally) or through the management interface (in the case of a timer event). The interpreter searches the event tables for an entry which matches the notification. If no match could be made, the notification is rejected. In the case of a message event, for example, this means that the corresponding operation is executed without changing the original parameters or taking any other management action. If the interpreter finds a relevant entry, the next step will be the evaluation of the corresponding condition in the table. The condition variable can either be an attribute or parameter. In both cases, it can be compared to an arbitrary value. If the condition is true, the action will be executed. After the action, the interpreter continues the search in the event table to detect if other rules match. Figure 5 shows the average delay caused by inspecting all entries of the rule table. The measurement has been made on a SUN IPC workstation using SunOS 4.1.3 and ANSAware 4.1 [14]. For further approximation of PCO delay or overhead, a linear dependency with respect to the number of rules can be assumed. The time for executing one or more rules is not included and has to be added for calculating the complete delay caused by a PCO.

3.6. Domain and policy service

Domain and policy services are fundamental for implementing policy-based management of distributed systems. The domain service is used to group objects together for management purposes and thus enables the manager to apply policies not only to individual but to sets of objects. The policy service is used for handling policy specifications and assigning them to domains. The basic concepts of domain and policy service are adapted from [3]. They are generic in the sense that they can be used for any kind of management application. Using our domain server, the manager

can create and delete domains, include domains into other domains and handle object membership. The policy identifier can be applied to domains, and operations are offered by which domain membership can be determined. The whole functionality enables the policy server to delegate the right policy to the right object without the need to define an extra policy for each object.

After defining a policy a manager translates it into rules using the policy server. Next, it assigns the rules to a certain domain. Therefore, the domain server is asked for the list of all domain members. The policy server then sends the rules to all objects in the list. This is the point where the PCO of the object starts working.

3.7. Example of a policy translation

As already mentioned in section 2.3 policy P3 allows the translation into rules. Therefore,

- the notification `NewJobArrived`, which is sent to the PCO by the message queue,
- the operation `RefuseJob` that enables PCO to refuse jobs arrived at the message queue,
- the notification `Alarm` forwarding a timer event to the PCO,
- the management attributes `QueueLength`, `Jobs-In Queue`, `Strategy`, `JobsRefused` and `Load-Report`,
- the generic actions `Get`, `Set` and `Inc` on attributes, and
- the notification `LoadReport` sent by the PCO to its manager

are used. On the basis of these operations, attributes and notifications, the policy is compiled into the following collection of rules:

```
E: Queue-->NewJobArrived(JID)
  C: Queue<--Get(JobsInQueue) = 20
    A: Queue<--Inc(JobsRefused,1)
      Queue<--RefuseJob(JID)
E: Server-->
  Threshold(CurrentLoad,>,0.8)
  C: Server<--Get(Strategy)=FIFO
    A: Server<--Set(Strategy,SEPT)
E: Server-->
  Threshold(CurrentLoad,<,0.75)
  C: Server<--Get(Strategy)=SEPT
    A: Server<--Set(Strategy,FIFO)
E: Timer-->Alarm(0900)
  C: true
    A: Server<--Get(LoadReport)
      Manager<--LoadReport(LoadReport)
```

3.8. Related work

The policy-based approach to network and systems management proposed by Wies in [15] and [16] defines a rich framework for management for policies, policy hierarchies and policy transformation. Policies are described by templates offering a large number of characteristics. On the lowest level policy goals are described by guarded commands. Although there exists

an implementation of a policy enforcement module, it is specific to HP Open View.

[17] propose to enrich managed objects with functionality as required by the management policy. The functionality describes the policy goal. Policies are divided into an active and a passive part. Whereas the passive part can be reused without change, the active part containing the functionality is application-specific. The approach presented in this paper follows the idea of having an application-independent behaviour description, that will be adapted to the local environment by the interpreter of the PCO. In addition, this approach is based on a different policy model containing no modalities. Every policy is assumed to be an obligation, i.e. a positive motivation policy. A prototype implementation has been realized using Smalltalk-80.

A similar approach based on enriching agents with functionality at runtime called ‘elastic processes’ has been proposed in [18]. It falls back on the ‘management by delegation’ principle that has been introduced by [19]. The Object Management Group is developing a management framework for their Common Object Request Broker Architecture (CORBA), see [20]. It makes use of domain and policy concepts. In contrast to the approach made in this paper, where policies can be added, modified and deleted at runtime, CORBA Management assumes all policies to be developed at design time. Therefore, it is less flexible.

In [21] an approach similar to the one presented above is used. It also enforces policies by means of rules, but the understanding of a rule is more restrictive. Again, all policies have a positive motivation modality. The policy management is implemented on top of the Spectrum Network Management Platform.

4. Performance measurement of a sample policy

4.1. The example scenario

We are modelling a server object with multiple interfaces, one for each client, see figure 6. A queue is associated with each interface in order to schedule the jobs of each client according to a certain policy.

This multi-interface server first has to be given a management interface. Therefore, we use a pseudo management interface description language. Interfaces are described by the rate of arriving jobs and the number of waiting jobs in the queue. The operation `Calculate RateDeviation` will return the largest deviation between all job arrival rates since the last time this operation has been invoked. This value is of major importance for the policy we are going to describe. The attribute `Count` sums the number of jobs in all queues. This attribute is attached with a threshold that emits a notification if 50 jobs have been queued since the last notification sending.

```
management interface
  MultipleInterfaceServer
types
  scheduling_policy =
  enum{
    Full.Queue.First,
```

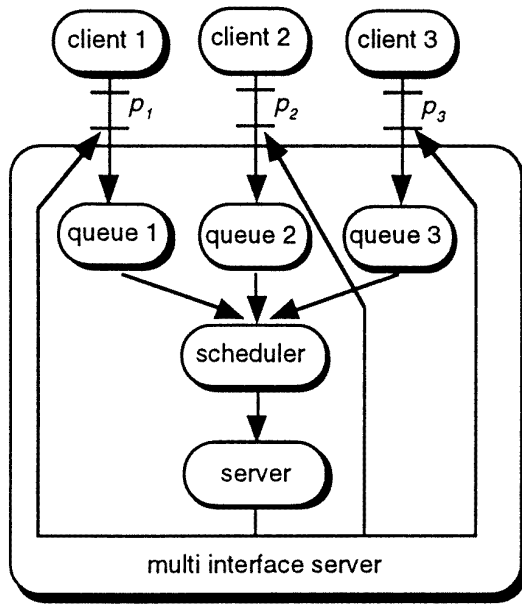


Figure 6. Multi-interface server scenario.

```

Longest_Queue_First,
Cyclic_Choice}
interface_desc =
record{
  rate:float,
  jobs_in_queue:cardinal}
attributes
  NumberInterfaces:cardinal: get;
  SchedulingStrategy:
  scheduling_policy: get, set;
  Interfaces:
  array[NumberInterfaces] <if_desc>:
  get, add, remove;
  Count: integer: get, reset;
operations
  CalculateRateDeviation
  (out: RateDeviation: float)
notifications
  CountThresholdReached:
  threshold <Count,50>;
end MI

```

4.2. From policy to rules

We now define an adaptive scheduling policy for the multi interface server based on the above described management interface. It defines the server manager, named `s_manager`, to be the policy subject and the server to be the object. The name of the policy is `Queue-SchedulingPolicy` and has only one defined behaviour called `AdaptiveScheduling`. It is possible to define further behaviour to be used instead. This policy is defined for all servers in the `SERVER` domain that are managed by members of the domain `Manager(SERVER)`. In general, PDN assumes each domain to have a manager domain. On reception of a threshold

notification related to the `Count` attribute the PCO requests the rate deviation to be calculated. Finally it changes the server's switching strategy depending on the current rate deviation value.

```

policy QueueSchedulingPolicy
for
  s_manager in Manager(SERVER) as manager
  and
  server in SERVER as managed
with
  behaviour AdaptiveScheduling is
  on notification CountThresholdReached
  at s_manager from server
  let rd :=
  GetRateDeviation(server.
CalculateRateDeviation()) in:
  force
  if (2<=rd<3)
  server.Set(SchedulingStrategy,
  Full_Queue_First)
  if (rd >= 3)
  server.Set(SchedulingStrategy,
  Longest_Queue_First)
  if (rd<2)
  server.Set(SchedulingStrategy,
  Cyclic_Choice)
  end behaviour
end policy

```

The `QueueSchedulingPolicy` is a positive obligation and will be transformed into a number of rules. In order to delegate the enforcement of the above policy, the server manager transforms it into a set of rules using the policy service. During this procedure, remote operation calls of the `s_manager` to the server like `server.Set(SchedulingStrategy,.)` will be replaced by local rules like 'Set STRATEGY .', where '.' represents an arbitrary scheduling strategy. The reception of a threshold notification is modelled by a threshold event on the attribute `Count`.

```

E: server<--Threshold(Count,eq,50)
C: true
A: server-->GetRateDeviation(RD)
E: server<--Threshold(Count,eq,50)
C: 2<=RD and RD<3
A: server-->Set(SchedulingStrategy,
  full_queue_first)
E: server<--Threshold(Count,eq,50)
C: RD>=3
A: server-->Set(SchedulingStrategy,
  longest_queue_first)
E: server<--Threshold(Count,eq,50)
C: RD<2
A: server-->Set(SchedulingStrategy,
  cyclic_choice)

```

4.3. Measurement results

To measure the gain achieved by using policy-based management in the scenario described above we used two

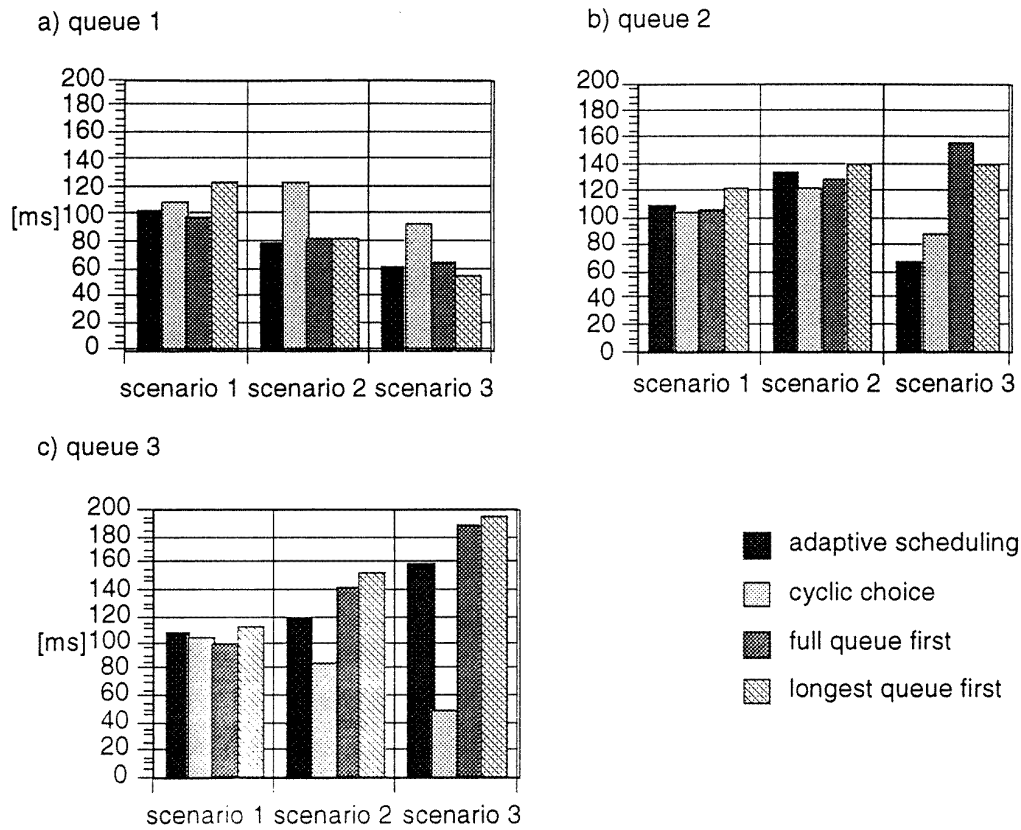


Figure 7. Average job waiting time for each queue in milliseconds.

Table 1. Measurement scenarios.

Arrival rate	p_1	p_2	p_3
Scenario 1	0.33	0.33	0.33
Scenario 2	0.5	0.3	0.2
Scenario 3	0.7	0.2	0.1

SUN IPC workstations using SunOS 4.1.3 and ANSAware 4.1 connected by an Ethernet LAN. All clients are placed on the same workstation whereas the server resides on the other one.

For the measurement experiments we consider three different scenarios for the distribution of jobs to clients. In scenario 1 all clients emit an equal number of jobs, whereas scenario 3 assumes a 7:2:1 relation.

Figure 7 shows the average job waiting time in milliseconds for all three queues. In four cases, the adaptive policy has advantages over the static scheduling policy. Only for the third queue having the least job arrival rate, is the Cyclic Choice strategy better. In order to minimize the job waiting time at all queues, the adaptive policy is optimal especially if we take into account that—depending on the scenario—queue 1 contains most of the jobs within the multi-interface server.

5. Conclusions

In this paper we presented an new approach to the enforcement of management policies. The key to our method lies in the transformation of policies described in a formal notation into a set of event-triggered rules. In addition, we have decentralized management behaviour by a concept called *delegation by policy* that means a manager delegates control to the agent by forwarding a policy. Each agent is given a Policy Controller Object (PCO) in order to interpret policies. Rules can be in three ways, by message events, threshold event and timer events. Once an event occurs an additional condition has to be checked before the action part of the rule will be executed. We have implemented a generic PCO for ANSAware objects and have measured the delay caused by the PCO within a remote service call. We have illustrated our method by giving an example for the management of a multi-interface.

Acknowledgments

This work is supported by the Deutsche Forschungsgemeinschaft (DFG) under Grant No Sp 230/8-2.

References

- [1] Sloman M (ed) 1994 *Network and Distributed Systems Management* (Reading, MA: Addison-Wesley)

- [2] Hegering H and Abeck S 1993 *Integrated Network- and Systems Management* (in German) (Reading, MA: Addison-Wesley)
- [3] Becker K, Raabe U, Sloman M and Twidle K 1993 (eds) *Domain and Policy Service Specification* IDSM Deliverable D6, SysMan Deliverable MA2V2, S-SI-07-I-2-R
- [4] ISO/IEC JTC1/SC21 10746-1 to -4 1995 *Open Distributed Processing—Reference Model Part 1: Overview, DIS, Part 2: Foundations, IS, Part 3: Architecture, IS, Part 4: Architectural Semantics, CD*
- [5] Moffett J and Sloman M 1991 The Representation of Policies as System Objects *Domino Report B1/IC/6.1*
- [6] Moffett J and Sloman M 1993 Policy hierarchies for distributed systems management *IEEE J. Select. Areas Commun.* **11** 1404–14
- [7] Wies R 1995 Using a classification of management policies for policy specification and policy transformation *Integrated Network Management* ed A Sethi, Y Raynaund and F Faure-Vincent (New York: Chapman & Hall) pp 44–56
- [8] Meyer B and Popien C 1995 Flexible management of ANSAware applications *Open Distributed Processing—Experiences with Distributed Platforms, ICODP'95* ed K Raymond and L Armstrong (New York: Chapman & Hall) pp 271–82
- [9] Meyer B and Popien C 1994 Defining policies for performance management in open distributed systems *Proc. 5th IFIP/IEEE Int. Workshop on Distributed Systems. Operations and Management (DSOM'94) (Toulouse, 1994)*
- [10] Dittrich K, Gatzju S and Geppert A 1995 The active database management system manifesto: a rulebase of ADBMS features *Rules in Database Systems (LNCS 985)* ed T Sellis (Berlin: Springer) pp 3–17
- [11] Paton N, Campin J, Fernandes A and Howard W 1995 Formal specification of active database functionality: a survey *Rules in Database Systems (LNCS 985)* ed T Sellis (Berlin: Springer) pp 21–35
- [12] Aiken A, Widom J and Hellerstein J 1992 Behavior of database production rules: termination, confluence, and observable determinism *Int. Conf. Management of Data, SIGMOD Record* **21** 59–68
- [13] Baralis E, Ceri S and Paraboschi S 1995 Improved rule analysis by means of triggering and activation graphs *Rules in Database Systems (LNCS 985)* ed T Sellis (Berlin: Springer) pp 165–81
- [14] Architecture Projects Management Ltd 1993 *ANSAware 4.1, Application Programmers Manual*
- [15] Wies R 1995 Policies in integrated network and systems management *Dissertation* University Ludwig Maximilian at Munich
- [16] Hegering H, Neumair B and Wies R 1996 A corporate operation framework for network service management *IEEE Commun. Mag.* **34** 62–8
- [17] Roos J, Putter P and Bekker C 1993 Modelling management policy using enriched managed objects *Integrated Network Management III* ed H Hegering and Y Yemini Y (Amsterdam: North-Holland) pp 207–15
- [18] Goldszmidt G and Yemini Y 1995 Distributed Management by Delegation *Proc. 15th Int. Conf. on Distributed Computing Systems* (Los Alamitos, CA: IEEE Computer Society Press)
- [19] Yemini Y, Goldszmidt G and Yemini S 1991 Network management by delegation *Integrated Network Management II* ed I Krishnan and H Zimmer (Amsterdam: North-Holland) pp 95–107
- [20] 1995 *OMG RFC Submission: Systems Management: Common Management Facilities* Volume 1, Version 2
- [21] Lewis L 1996 Implementing policies in enterprise networks *IEEE Commun. Mag.* **34** 62–8