Defining Policies for Performance Management in Open Distributed Systems

B. Meyer*, C. Popien

Aachen University of Technology, Department of Computer Science IV
Ahornstr. 55, D-52056 Aachen, Germany
Phone: +49(241)8021440, Fax: +49(241)8888220
E-Mail: bernd@informatik.rwth-aachen.de

Abstract
An important issue for managing distributed systems is achieving a high flexibility of management components. A means to increase flexibility of management systems is defining a policy for the managers behaviour. A policy determines the behaviour of an object and is enforced by a manager (or administrator) object. In this paper, a performance policy for a trader policy domain is defined. It is deduced from results obtained by an analytical tool. Therefore a system model of a distributed system is constructed using extended fork join nets. The policy is specified using the policy definition notation presented in this paper,. It is a common language for policy specification and can be used in various areas of management.

1. INTRODUCTION

In the last decade, an evolution from large mainframe computers to networks of interconnected workstations has become obvious. At first, an open interconnection between computers of different vendors using different operating systems without regard of the underlying network architecture and technology was needed. With increasing computing power of workstations, the question for open distributed processing came up. It is intended to hide certain aspects of the distributed nature of a system from application programmers. Analogous to this development, most concepts developed for management have a special focus on administrating computer networks. Although basic management principles can be reused for managing distributed systems, there are certain aspects, that need to be included. On the one side system computing resources, see [ODP P3], and on the other side service or applications must be managed. Especially heterogeneity and autonomy of components of a distributed system require a high flexibility. In order to achieve a more flexible manager behaviour, it is determined by a policy. The manager must be able to interpret a given policy and start an activity to enforce the desired behaviour on the involved objects. Flexibility of a manager depends on the different kinds of behaviour it is able to interpret. In the case that a sufficient large library of manager behaviour is available, a user of the management system might change the policy of a manager. A change of policy can be required by after integration of new components into the distributed system, especially if it has a local management system of its own. Another possibility is, that the policy depends of a number of system parameters, that are monitored from time to time.

In the last view years, several approaches were started to achieve openness in distributed processing. Some platforms like the Distributed Computing Environment (DCE) [Sch92] of the Open Software Foundation (OSF) and the Common Object Request Broker Architecture (CORBA) [OMG] of the Object Management Group (OMG) and the Reference Model for

* This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under Grant No. Sp 230/8-1
Open Distributed Processing (RM ODP) [ODP P1-4] of the ISO were introduced to cope with these problems. One important tasks of ODP is managing services offered in a distributed system. Service Management is done by the ODP Trading Function [ODP TF] and is realized by an object called the ODP Trader. It stores services exported to the trader in a trader database (TDB), where it can be referenced for service imports. Much work for designing distributed management systems could be saved, if it is based on one of these distributed computing platforms. Since ODP is the most general approach and is defined independently of vendor policy, it is chosen in the presented approach.

In section two Performance Management is chosen to serve as an example for the application of policies. The performance manager is supported by a tool for the analytical performance evaluation of distributed systems. It enables a manager to compute the average response time of a request in a distributed system, where concurrent processing is possible. The following section introduces the Policy Definition Notation (PDN), a formal notation for specifying management policies in distributed systems. The syntax of this notation is described by using EBNF. Section four uses the ODP Trader to give an example, how performance requirements can be derived from analytical performance evaluation results. In addition, it will be shown, how the PDN defined in section three can successfully be used for policy specification.

2. USING POLICIES FOR PERFORMANCE MANAGEMENT APPLICATIONS

In this section we choose the area of Performance Management for the application of policies. It is concerned with determining the quality of service parameters, monitoring a system in order to detect performance bottlenecks, storing measured parameter values, creating performance reports and planning performance and capacity of a system, see [HeAb93]. Examples for relevant parameters are load, throughput and response time of network connections or systems resources. In addition, it is interesting to monitor and store the number and size of requests to achieve a base for analysis or simulation of a system. The results can be used in order to improve system performance.

![Figure 1: Algorithm for Performance Management](image)

The function of Performance Management consists of four tasks, which are modelling the system, monitoring or measuring performance parameters, computing target parameters and finally tuning the system. For achieving an acceptable reaction time of the performance management application, the choice of method for computing target values is important. For time consuming approaches, the target value calculation should be done once in advance or
when the system load is low for a longer time. As an alternative, an approximative calculation of upper or lower bounds of the target value should be considered. After constructing a system model, system parameter values are to be monitored, see figure 1. Now, policies can be deduced with the help of new values of the target parameter. The actions starting with monitoring system parameter values are repeated after a given amount of time or on a notification from corresponding monitoring managed objects.

For calculating the average response time, the approach presented uses an analysis tool, see [MePo93b]. In contrast to traditional operating systems, where every computer has manages its own resources, in a modern system resources are distributed and requests to use them might be processed concurrently. Consistently, models for system analysis must also support concurrent or parallel task processing. Unfortunately, traditional queueing systems theory does not support parallel task processing. In the last decade, queueing models were extended to fork join nets enabling the analysis of parallel tasks, see [DuCz87]. Fork join nets must be further extended to allow request, that do not necessarily use all components of a distributed server, see [MePo93b], to meet requirements of distributed systems completely.

![Figure 2: Example for a model of a distributed system server](image)

The queueing model of a distributed system server, as shown in figure 2 for two server, consists of a number of distributed server offering a single logical system service. Each user request is forwarded to a subset of all server in the system, processed locally at these server stations. After computing its part of the result, it is sened back to the requestor. System parameters of this model are the number of servers, the arrival rate at each server, the service rate at each server and the number of servers involved in one request. At the current state, the analysis tool computes the average response time of server requests.

3. PDN - A FORMAL NOTATION FOR MANAGEMENT POLICIES

Referring to the RM ODP, a policy is defined as a prescriptive relation between one or more objects and some reference behaviour. It can be expressed as an obligation, a permission and a prohibition. By definition, behaviour is expressed as a collection of actions.

```plaintext
<policy> ::= "policy" <policy_name> "for" <policy_target> "with"
 ( <policy_behaviour> ["where"
 ( <behaviour_specification>) ] ) "end policy".
<behaviour_specification> ::= <policy_behaviour> | <informal_spec>
```

Within PDN, a policy consists of a unique name, a specification of the policy target and the corresponding behaviour. If the policy behaviour contains names of additional behaviour specifications, they have to be specified using the where statement. These behaviour specifications can consist of further policy behaviour or of an informal specification. An informal specification can be sensible, because it is not the task of a policy to prescribe computational behaviour in detail. Nevertheless behaviour can be split into blocks increasing readability of specification and helping the improve specification structure.
Policy_target describes the objects involved in this policy. A policy can be bound to one or a number of interacting objects. If it is bound to only one object, all observable behaviour specified in the policy is related to the object’s interaction with any other object. It is possible to specify an object enforcing the policy. This object is acting as a manager or an administrator of the other objects. From a more general point of view, single objects are often assumed to include components for enforcing policies, whereas objects at a lower level of abstraction are supposed to interact with manager objects enforcing policies. Within PDN objects are specified by an identifier (written in small letters), by an identifier for a enumeration of identifiers (first letter capital) or an identifier for a class (capital letters), see production rule for object_spec. If a set or a class of objects is specified, the policy should be valid for all members. The example TraderUserPolicy defines the behaviour ExportOnly, that only allows export, withdraw or replace operations to be called at the interface of trader_a. It is not determined, who is going to enforce the policy - trader_a or any object accessing it. In example TraderFederationPolicy, trader_a is chosen for policy enforcement. It enables trader_b to access the interface of trader_a to call export, withdraw or replace operations.

```
policy TraderUserPolicy
for trader_a
with behaviour ExportOnly is
    restrict interface to export, withdraw, replace
end policy

policy TraderFederationPolicy
for trader_a and trader_b enforced by trader_a
with behaviour DirectedFederation is
    restrict interface of object trader_a to export, withdraw, replace
end policy
```

Another criteria for classifying policies is the used level of abstraction. It depends on the object model used for policy definition. One model could regard an object as a black box. All of the object’s behaviour is described in terms of observable actions occuring at the interface of an object. Actions can be operations and notification, given by a signature including name and attributes. Internal object state does not matter in this viewpoint. This view is common to specification languages for distributed systems, for example the Formal Description Technique (FDT) LOTOS. For defining policies it is very useful to describe state-dependent behaviour. Therefore, the used object model has to contain the object’s state. The state of an object is given by its structure, e.g. the attributes an object is build of, and the current values of these attributes. The approach taken in this work is an object model including interface and state of an object. The interface is described by operations and notifications and its attributes are defined by type and name. The PDN production rules enable the description of an interface and its subparts (item_desc) and the corresponding values (item_spec). In terms of Network Management, we assume a Managed Object to be defined for each object involved in a policy.

```
<item_desc> ::= <behaviour_desc> | <interface_desc> | <attribute_desc>.

<item_spec> ::= <behaviour_spec> | <interface_spec> | <attribute_spec>.
```
A policy behaviour is identified by an unique name. Its behaviour can be expressed in terms of an obligation, a permission, a prohibition (see policy_modality), a disjunction or conjunction of behaviour expressions or a name representing a behaviour specification defined later on. In addition, behaviour can depend on the occurrence of an event or a condition.

A certain behaviour can be assigned to an object by a policy, whereas changes of an object's behaviour can be triggered by certain event occurrences or by certain conditions. Receiving an operation invocation or a notification are candidate events for changing object's behaviour. A change in behaviour can also be triggered by a certain object state.

An obligation specifies a behaviour objects must fulfil, contrary to a prohibition, that specifies an obligation forcing a certain behaviour not to occur. A permission allows a certain behaviour to occur, but it does not force its occurrence. The semantic of a permission is defined as a behaviour, that is not prohibited to occur. Hybrid behaviour can be specified using the restrict statement. It permits certain values for object state and subparts of object interface to occur, while all other values are prohibited. It is out of the scope of this paper to present formal semantics of these expressions. At least, it should be noticed, that they include modal logic operators. The semantics of these statements should be clear as far as their exact meaning is necessary for understanding of this paper.
The following example defines one aspect of the TraderSearchPolicy, see section 4. It is related to the consistency requirements for dynamic attributes of a service offer. Consistency can take the values low, medium and high. For low consistency requirements the trader can decide how often it will update the value of an attribute, whereas medium requires a periodic update of attribute values. If an attribute value has to be as consistent as possible with the actual value stored by the exporter, every change must be forwarded to the trader.

```
policy TraderSearchPolicy for trader_a and Exporter := {exporter_a, exporter_b} enforced by trader_a with
behaviour AdaptiveAttributeValueUpdate for trader_a is
  force
    on operation invocation SetConsistency with NewConsistencyValue =>
      if (NewConsistencyValue = low )
        then behaviour UpdateOnDemand for Exporter
      if (NewConsistencyValue = medium )
        then behaviour PeriodicUpdate for Exporter
      if (NewConsistencyValue = high )
        then behaviour UpdateOnChange for Exporter
  where
    behaviour UpdateOnDemand for Exporter is …
    behaviour CyclicUpdate for Exporter is …
    behaviour UpdateOnChange for Exporter is …
end policy
```

4. SPECIFYING PERFORMANCE REQUIREMENTS OF THE ODP TRADER

Service Management within ODP is done by the Trading Function, see [PoKu94]. It is realized by an object called the ODP Trader. In order to advertise a service, an exporter forwards a service offer, see figure 3. The Trader stores the offer in its TDB. If an importing object asks for a certain service, the trader looks up its TDB for an appropriate service offer. If a matching service offer can be found, the trader returns an interface reference for the server object offering this service.
If the search in the local TDB is not successful, a trader can establish remote access to another traders database. All traders associated with one administrator and their users are called a trader policy domain. It is determined by the trading policy, which traders allow remote access to their TDB. Consistently, to establish a link for remote access, a trader must contact its administrator, see [MePo93a]. Once the link has been established, it can be used until it is released by the other trader or the administrator. The main activities within the Trading Function are service import and service export. To govern these activities, a trading policy defines certain rules for behaviour specification. All traders, exporters and importers are obliged to obey these rules. The most important requirements specified by policies are:

- security requirements for importers and exporter specifying rules to prevent unauthorised use, disclosure and modification of service offers,
- searching requirements defining search space and strategy for service import,
- accounting requirements determining charges for service import and export, and additional functions offered by the trader,
- transfer requirements allowing importers to pass imported service offers to other users or exporters to export service offers of other users and
- quality of service requirements describing rules for performance requirements for service offers and trader service, consistency requirements for dynamic attributes.

One subpolicy of the trading policy, the trader placement policy, is used in the following example. It determines, at which trader a recently advertised service offer should be placed. For determining a policy, a minimal interface or managed object must be available. In the following example, a policy for the trader object should be developed. First, an extract of a managed object of the trader is defined.

**Managed Object** Trader

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDB_Size</td>
<td>Get</td>
</tr>
<tr>
<td>MaxTDB_Size</td>
<td>Get, Set</td>
</tr>
<tr>
<td>ServiceRate</td>
<td>Get</td>
</tr>
<tr>
<td>WaitingQueueLength</td>
<td>Get</td>
</tr>
<tr>
<td>MaxWaitingQueueLength</td>
<td>Get, Set</td>
</tr>
<tr>
<td>AverageResponseTime</td>
<td>Get</td>
</tr>
<tr>
<td>MaxAverageResponseTime</td>
<td>Get, Set</td>
</tr>
</tbody>
</table>

**Notifications**

- TDB_Updated
- TDB_Overflow
- WaitingQueueFull
- AverageResponseTimeOverflow

**EndMO**

For the use in the following example, only the size of the TDB, determined by the number of service offers stored, and the average response time are of concern. If the value of each attribute reaches a certain upper bound, that is define by the attributes MaxTDB_Size respectively MaxAverageResponseTime, a notification is sent to the manager object. In addition, a notification is sent, if the size of the TDB is changed. For placing service offers at
a certain trader, there are several strategies available. The random-placement strategy takes any trader in the trader policy domain and places new service offers (user requests) there, whereas the local-placement strategy uses the trader located at the same computer node. Cyclic-placement stores all traders in a cyclic list and places each service offer at the next trader in the list, whereas place-at-lightest-trader looks for the trader, that has the smallest TDB with respect to its processing power. Therefore it needs the length of the current waiting queue and the service rate of all traders in the domain. Finally, the adaptive-placement strategy switches between a choice of these strategy.

![Figure 4: Average response time depending on TDB size and placement strategy](image)

The analysis of a system model calculating the average response time of trader request has been done for two service offer placement strategies, the local-placement strategy and the place-at-lightest-load-trader strategy, see figure 4. For achieving a minimal average response time, the placement strategy for the trader domain must be determined depending on the number of service offers of the requested trader. As result, the following trader placement policy can be deduced.

```plaintext
policy TraderPlacementPolicy
  for TRADER enforced by trader_administrator with
  behaviour AdaptivePlacement for trader_administrator is
    force
      on TDB_Updated from TRADER with TDB_Size =>
        if (SizeTDB in [0, 320) )
          then behaviour LocalPlacement for TRADER
        if (SizeTDB in [320, 550) )
          then behaviour PlaceAtLightestLoadTrader for TRADER
        if (SizeTDB in [550, 600) )
          then behaviour LocalPlacement for TRADER
    where
      behaviour LocalPlacement is ...
      behaviour PlaceAtLightestLoadTrader is ...
  end policy
```
The adaptive-placement strategy falls back on the local-placement and place-at-lightest-load-trader strategy, which are not further specified here. The policy defined above must be enforced by the trader administrator as the manager of all traders in the domain, since knowledge of load of all (or at least a subset of all) traders in the domain is necessary for deciding with the place-at-lightest-load-trader strategy.

5. CONCLUSIONS

In distributed systems, components of the management system must provide a high level of flexibility. The approach presented in this paper realizes flexibility by determining a policy for manager behaviour. A performance policy for service trading in open distributed systems is deduced from the results of an analysis of a distributed system. The distributed system is modeled by a queueing model, an extended fork join net. With the help of an analysis tool, the average response time of requests is calculated. From the results, a policy is deduced using a newly developed definition notation, the policy definition notation. Although only used for performance management within this paper, the PDN serves as a general notation for any policy used by a management system.

Further work on PDN will include defining formal semantics for policies. If more than one policy is defined for one objects, conflicts between them are likely to occur. To solve these conflicts, intersection and union of policies must be defined. Another important issue is finding an appropriate language for behaviour description. It must be checked whether the Formal Description Techniques (FDT) SDL, LOTOS, Estelle [Tu93] or other specification languages, e.g. Z, are good candidates. It has to be suitable for stepwise refinement, so precise behaviour specification can be deferred until object interfaces are determined. Consistently, defining and refining policies is an iterative process and will be used throughout analysis and design phase. Speaking in ODP terms, the enterprise, information and computational viewpoints are relevant, see also [PuRo91].

REFERENCES

<table>
<thead>
<tr>
<th>Reference</th>
<th>Source</th>
</tr>
</thead>
</table>