Project Number :	**IST-1999-10561-FAIN**
Project Title :	**Future Active IP Networks**

Initial Specification of Case Study Systems

Editor :	**Célestin Brou**
Document No:	**WP4-GMD-011-D3-Pub**
Contribution File Name :	**WP4-D3v1.doc**
Version :	**1.0**
Company :	**GMD**
Date :	**Tuesday, May 29th 2001**
Distribution :	**Pu**
Dissemination:	**Pu**

The FAIN Consortium consists of:

Partner	Status	Country
UCL	Partner	United Kingdom
JSIS	Associate Partner to UCL	Slovenia
NTUA	Associate Partner to UCL	Greece
UPC	Associate Partner to UCL	Spain
DT	Partner	Germany
FT	Partner	France
KPN	Partner	Netherlands
HEL	Partner	United Kingdom
HIT	Partner	Japan
SAG	Partner	Germany
ETH	Partner	Switzerland
GMD	Partner	Germany
IKV	Associate Partner to GMD	Germany
INT	Associate Partner to GMD	Spain
UPEN	Partner	USA

The FAIN Consortium

University College London	(UCL)
Josef Stefan Institute	(JSIS)
National Technical University of Athens	(NTUA)
Universitat Politecnica De Catalunya	(UPC)
T-Nova Deutsche Telekom Berkom GmbH	(DT)
France Télécom / R&D	(FT)
Koninklijke KPN NV, KPN Research	(KPN)
Hitachi Europe Ltd.	(HEL)
Hitachi Ltd.	(HIT)
Siemens AG	(SAG)
Eidgenössische Technische Hochschule Zürich	(ETH)
GMD Forschungszentrum Informationstechnik GmbH	(GMD)
IKV++ GmbH Informations- und Kommunikationstechnologie	(IKV)
Integracion Y Sistemas De Medida, SA	(INT)
University of Pennsylvania	(UPEN)

Project Management

Alex Galis
University College London
Department of Electronic and Electrical Engineering,
Torrington Place
London WC1E 7JE
United Kingdom
Tel +44 (0) 207 458 4463
Fax +44 (0) 207 388 9325
E-mail: a.galis@ee.ucl.ac.uk

Authors

Celestin Brou (GMD) – Editor
Hui Guo (GMD)
Richard Sinnott (GMD)
Mehran Roshandel (DT)
Matthias Bossardt (ETH)
Bertrand Mathieu (FT)
Christos Tsarouchis (HEL)
Chiho Kitahara (HIT)
Jürgen Dittrich (IKV)
Juan Luis Manas (INT)
Franci Mocilar (JSIS)
Jan Laarhuis (KPN)
H.C.G Pragt (KPN)
Yiannis Nikolakis (NTUA)
Odysseas Pyrovolakis (NTUA)
Ermolaos Zimboulakis (NTUA)
Alex Galis (UCL)
Alvin Tan (UCL)
Kun Yang (UCL)
Epi Salamanca (UPC)
Joan Serrat (UPC)
Julio Vivero (UPC)

Abstract

D3 is a deliverable composed primarily of the internal reports R11, R12, R13 and R25. The intention of this deliverable is to provide an initial specification of the case studies that will be used to demonstrate the realisation of the architecture and overall general approach of the FAIN project. This deliverable itself provides an overall Policy Based Active Network Management Architecture (PBNM, PBANEM) along with an integrated Active Service Provisioning (ASP) Architecture that has been aligned with the FAIN Enterprise model. Sample scenarios are provided which are being used for (ongoing) prototyping of the underlying architecture and approach of FAIN. It is likely that these scenarios and prototypes will be used as the basis for further areas of implementation in FAIN, as and when the FAIN architecture and methodology is further refined.

Keywords

Active Network, Active AN Management, Active Service Provisioning,, XML, Policy Based Management, Agent Technology, DPE, QoS Delegation.

Change History

Version	Author	Comments
V0.0	WP4 Workgroup	Initial Specification of Case Study Systems
V1.0	WP4 Workgroup	Integration of all comments.

ACRONYMS

AC: Active Code

AN: Active Networks

ANE: Active Network Element

ANN: Active Network Node

ANSP: Active Network Service Provider

API: Application Programming Interface

ASN: Abstract Syntax Notation

ASP: Active Service Provisioning

BML: Business Management Layer

CDB: Conflict Detection Block

CIM: Common Information Model

CLI: Command Line Interface

CMIP: Common Management Information Protocol

COPS: Common Open Policy Service

CORBA: Common Object Request Broker Architecture

DAP: Directory Access Protocol

DCE: Distributed Computing Environment

DCN: Data Communication Network

DEN: Directory Enabled Networks

DIT: Directory Information Tree

DME: Decision Making Entity

DMTF: Distributed Management Task Force

DPE: Distributed Processing Environment

DSCP: Diffserv Code Point

EE: Execution Environment

EM: Element Management

EMS: Element Management System

FAIN: Future Active IP Networks

FAIN TA: FAIN Technical Annex

FCAPS: Fault Configuration Accounting Performance Security

FIFO: First In First Out

GDMO: Guidelines for Definition of Managed Objects

GUI: Graphic User Interface

IDL: Interface Definition Language

IETF: Internet Engineering Task Force

ISE: Information Storage Entity

ITU: International Telecommunication Union

JDBC: Java Database Connectivity

JNDI: Java Naming and Directory Interface

LAN: Local Area Network

LDAP: Light Directory Access Protocol

LPDP: Local Policy Decision Point

LRU: Least Recently Used

LSP: Label Switched Path

MA: Mobile Agents

MD: Mediation Device

MF: Mediation Function

MIB: Management Information Base

MIF: Management Information Format

MPLS: Multiprotocol Label Switching

NACK: Not Acknowledged

NE: Network Element

NEF: Network Element Function

NIP: Network Infrastructure Provider

NM: Network Management

NMF: Network Management Forum

NMS: Network Management System

ODBC: Open Database Connectivity

OMG: Object Management Group

ORB: Object Request Broker

OSD: Open Software Description

OSF: Operating System Function

PBANEM: Policy-based Active Network Element Management

PBANM: Policy-based Active Network Management

PBM: Policy-based Management

PBN: Policy-based Networking

PBNM: Policy-based Network Management

PBVPN: Policy-based Virtual Private Network

PCIM: Policy Core Information Model

PCIMe: Policy Core Information Model extensions

PDP: Policy Decision Point

PEP: Policy Enforcement Point

PHB: Per-hop Behaviour

PIB: Policy Information Base

QAF: Q Adaptor Function

QoS: Quality of Service

QPIM: QoS Policy Information Model

RAP: Resource Allocation Protocol

RCF: Resource Control Framework

RDBMS: Relational Database Management System

RSVP: Resource Reservation Protocol
SC: Security Context

SID: Security ID

SLA: Service Level Agreement

SML: Service Management Layer

SNMP: Simple Network Management Protocol

SP: Service Provider

SPPI: Structure of Policy Provisioning Information

SQL: Structured Query Language

SSL: Secure Sockets Layer

TCA: Traffic Control Agreement

TINA: Telecommunications Information Networking Architecture

TMF: Telecommunications Management Forum

TMN: Telecommunications Management Network

TOM: Telecom Operations Map

ToS: Type of Service

TTCN: Tree and Tabular Combined Notation

UML: Unified Modelling Language

VE: Virtual Environment

VPN: Virtual Private Network

WAN: Wide Area Network

WSF: Workstation Function

XML: Extensible Markup Language

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The Deliverable D3 describes the initial implementation of case study systems in the FAIN project. It is composed primarily of internal reports R11&R12 (Fain initial Policy Based Network management), R13 (Active Service Provisionning) which are outlined in chapters 2 and 3 respectively. Report R25, the Initial Summary Networks Management Issues for Active Networks are added as an Appendix. The context of Network Management and Active Service provisioning in FAIN along with assumptions on the architectural choices are explained in the associated chapters. The document focuses on the design of architectures and presentation of candidate scenarios to be implemented to demonstrate and justify the overall FAIN approach.

2 ACTIVE POLICY BASED MANAGEMENT FOR ACTIVE NETWORKS

In this chapter management aspects in relation to active networks are studied and described. The management concepts and approaches described in this deliverable are initial ones which will be iteratively refined and improved throughout the life of the project.

This chapter is structured as follows. Chapter 2.1 provides the context of the management activity within FAIN. Chapter 2.3 presents an initial analysis of the FAIN management requirements and chapter 2.4, presents the initial FAIN management architecture. The application and applicability of this initial architecture to the different management levels is described in Chapter 2.5. An initial, implementation of part of this architecture is presented in Chapter 2.6. Finally in Chapter 2.7 conclusions on the initial effort on management are given and recommendations and issues for improvement and extension of the initial architecture and implementation are described

2.1 INTRODUCTION

The main purpose of this introductory chapter is to provide a careful explanation of active policy based management for active networks and describe how the FAIN project proposes to deal with management aspects as they pertain there.

The scope of the management research within FAIN is described in Section 2.1.1 with arguments given for using active technology for management purposes in Section 2.1.2. Then an overview of the current state of the art in active approaches for management is given in Section 2.2.3 with section 2.2.4 focusing on policy-based management approaches in particular. The high level approach to the management system for FAIN is described in Section 2.2.5 and concludes this introductory chapter.

2.1.1 Scope and Objective

Within FAIN we confine ourselves to the two lowest management levels of the Telecommunication Management Network (TMN) model: network element management and network management. These levels are in line with the main overall objective of FAIN: designing active nodes and building active networks from them. In addition to these two levels the interface with the service level is studied and described.

The title of this chapter needs to be elaborated in order to understand the general direction of the management work to be undertaken within FAIN. Leaving out, in first instance, the management style, i.e., 'policy-based', the title reads 'active management for active networks'. This contains two aspects when we leave out one of the words 'active'. The first is 'management of active networks' meaning that active networks have to be managed. The second is ' active management of networks' meaning that the intention is to useing active technology for management purposes. Obviously, the ultimate goal in FAIN is the combination of both, as expressed by the title, while the ultimate management style is policy-based. This is where we want to go with management in FAIN, i.e., and the target situation. The current situation for management can predominantly be described as 'non-active SNMP-based management of non-active networks'. The target situation thus differs from the current situation in two main respects: the implementation (active or non-active) of both the management and data plane, and the management style used (SNMP-based or policy-based). Leaving out the management style, there are two paths to reach the target situation from the current situation (see Figure 1).

The upper path in Figure 1 entails an 'activation' of the management plane first. As a consequence, it means applying active technologies and capabilities to the management plane, which serve to manage traditional networks initially. Thus the upper path leads via active management of traditional nodes (routers, switches) to the target situation. The lower path in Figure 1 entails an 'activation' of the data plane first. This means that non-active management or a certain style is applied to active nodes and active networks. Thus the lower path leads via non-active management of active networks to the target situation.

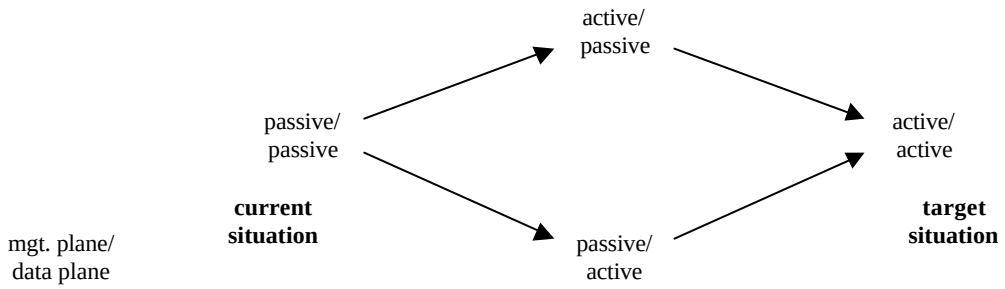

Figure 1 – Possible management migration strategies

In FAIN we will pursue both paths towards the target situation for several reasons.

The lower path will be pursued within FAIN from the outset with a policy-based management style. In fact, this path entails bringing active nodes and active networks under the regime of policy-based management. The most important advantage of this approach is the increased flexibility in management offered by the policy-based style. This flexibility is needed to properly manage and control the enhanced capabilities, w.r.t. flexibility, of active nodes and active networks. However, notice that following this path requires as a prerequisite, early availability of the FAIN active node. Based on this observation, and the fact that both policy-based and active network-based techniques are emerging technologies, the advantages of this lower path will be available on a longer term.

The upper path offers several advantages. In the first place, it has been proven in the past (see Section 2.2.3.1) that deploying active technology in the management plane leads to more effective and more efficient management. Secondly this path provides a smooth evolution towards the target situation because the FAIN active node only needs to be available in the final step towards the target situation. It therefore respects the design time of the FAIN active node. Moreover, by following the upper path, it is expected that vendors and operators will benefit from active technology in the short term. And that because application of this technology to the management plane puts less constraints on performance and security (only one user, i.e., the operator!) and thus on the maturity of the technology. A second and later step in the upper path is to enhance the system with a policy-based management style.

In the target situation the advantages of both paths are combined. This is expected to happen in the final stage of the project and includes management of the active nodes developed within the project by WP3.

2.1.2 FAIN Overall Network Management Environment

The FAIN active node architecture defines active nodes, which provide full flexibility to the user to manage and provide active services.

The defining characteristic of an active node is the ability for users to load software components dynamically and offers dynamic programmability. An active node provides at least an execution environment, which provides the capability of running user-provided code. An important feature of active nodes is that any undesirable operation will have limited consequences. This can be achieved safely since each customer can in principle be provided with a dedicated active node and that customers who are sharing the same active node would be provided with a partition of the node resources via a private VPN.

Packets requiring active processing are marked in some way to allow correct handling by active routers. This allows the discrimination of active and conventional packets and the selection of an active node. Routing and node resources configuration in the active nodes could be achieved by setting policies at the network management level. The management approach in the FAIN project is based on policies.

We envisage that the management of the active network will require:

- **Policies:** Design of management policies required to manage the active nodes and network
- **Management Components in the Active Nodes:** Design of management components for the active nodes, which will execute policies within an active node and which will monitor the node resources usage. The execution of policies means mapping target policies into node resource configurations
- **Management Nodes:** A set of management nodes that will provide mechanisms to enable network administrators to manage the active networks as a whole, including network policies set-up and processing.

The management components within the active node and the functions performed by these will determine the management capability of the active node and hence the extent to which management can be delegated to the node. The policy based management solution adopted by FAIN will serve to delegate management through the execution of policies. In this way, the policies will define the rules that will determine the behaviour of the active nodes. The execution of these policies will cause changes in the node and will determine how it delivers the services for users. The policies are defined and sent to the nodes by the management nodes, which deliver services. The management nodes will need to know what policies to send to which active nodes, when to send these and what results these have.

The network and service provider will need to be provided with management capability at the management nodes to monitor and control the whole and/or parts of the network.

The Network and/or Service providers will need to ensure the network resources are used efficiently and that enough resources are provided to meet users' demand. The Network and/or Service Providers will need to know if the active nodes are executing the right policies and that the active nodes are delivering the services correctly to users. As the delivery of services will require co-operation of a number of active nodes the network providers will need the means of managing the active nodes as a group of nodes and not individual nodes. They will need monitoring mechanisms for checking that correct policies are being defined and used in relation to the network before they are sent to the actual network. It will need to know what policies are currently loaded in the active nodes and what impact these are having on the network. It will also need to protect and monitor the security of the network. Therefore, the network/service provider needs a set of management mechanisms that will enable it to manage the network as a whole.

In FAIN we envisage that two types of management nodes provide these mechanisms:

- Element Management Nodes
- Network Management Nodes

The main difference in functionality provided by these two types of management nodes is in the policy types, which they could process and manage, in the sub-networks, which they cover and in the creation of management domains for different types of users as shown the following figures:

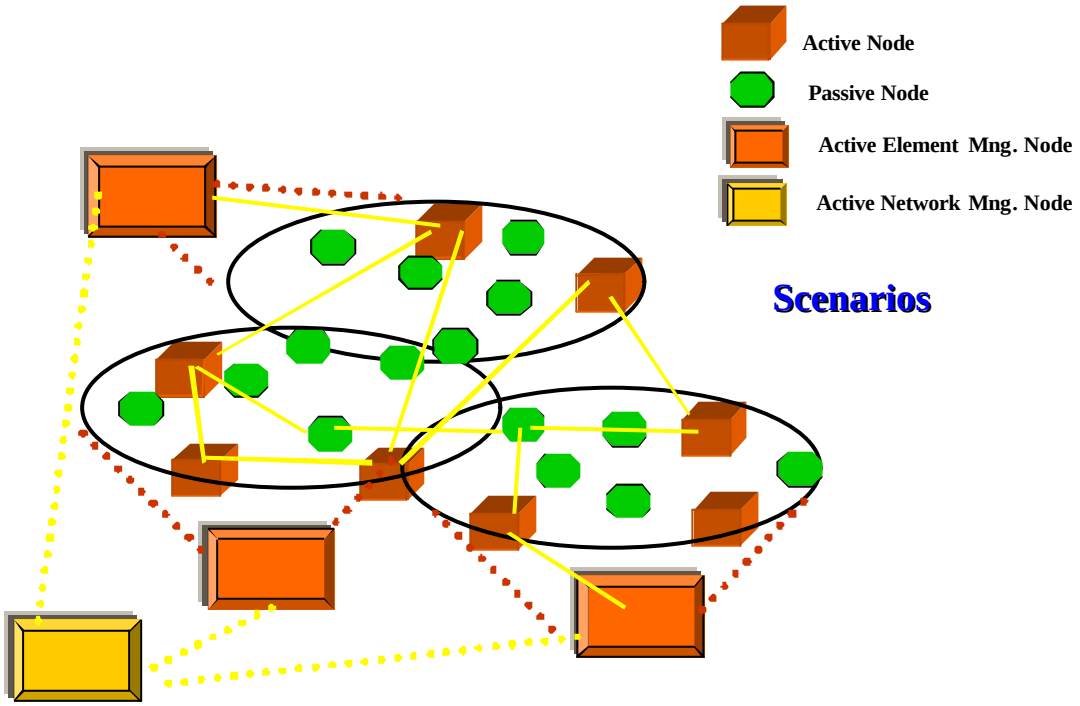

Figure 2 - Active Network Management

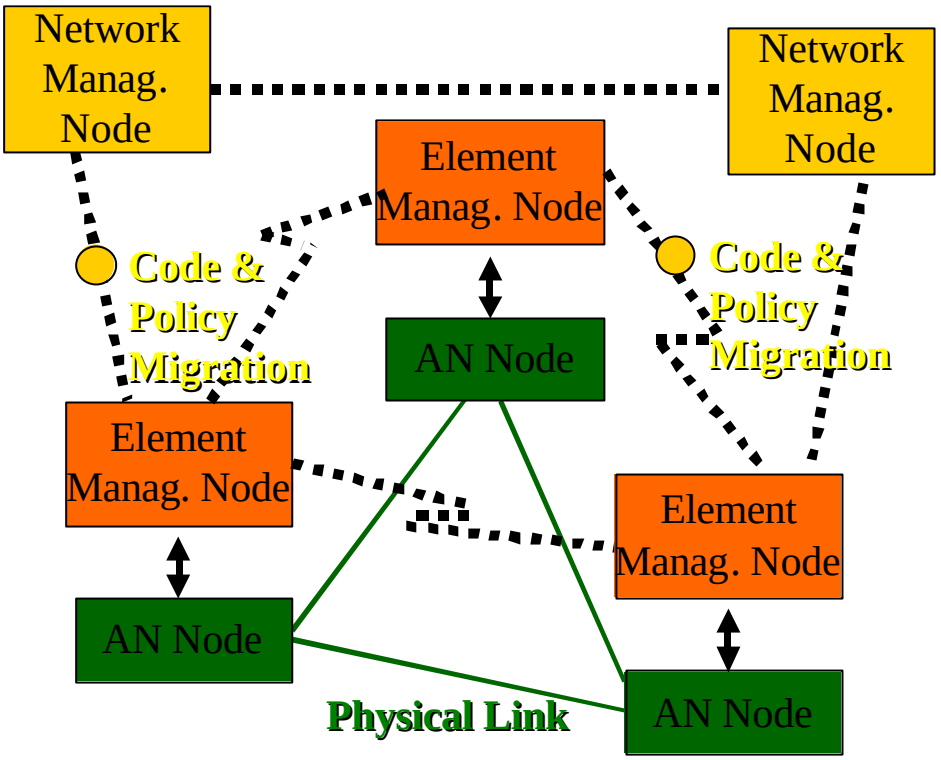

Figure 3 - Active Management Policies Migration

The relationships between **Active Management Nodes**, active **Management Components** in the active nodes and the **Policies** are shown in Figure 4.

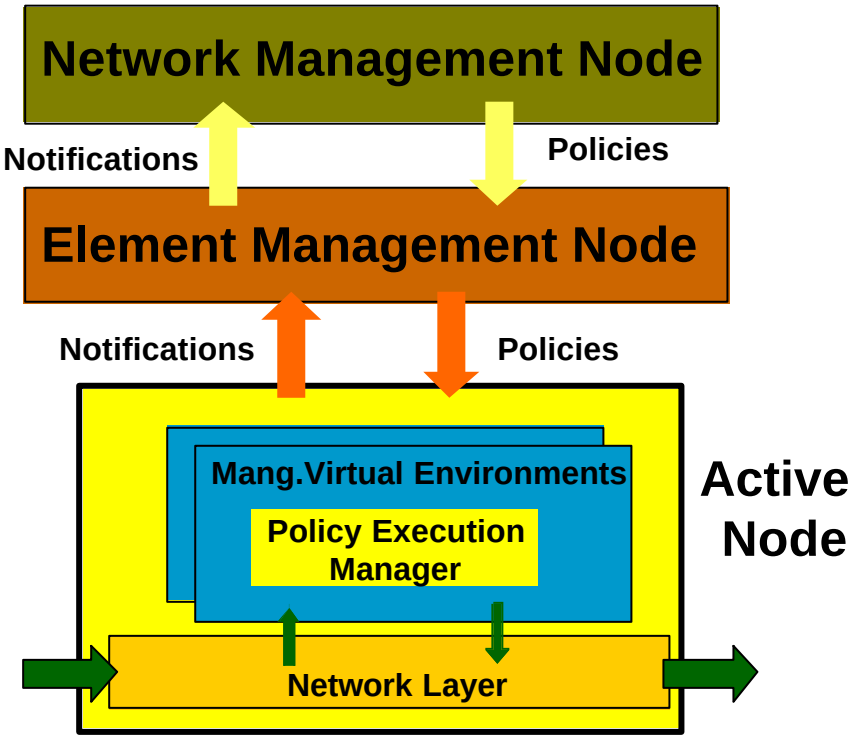

Figure 4 – Active Network Management

2.2 R11 - RATIONALE FOR USING ACTIVE NETWORKING APPROACH TO MANAGEMENT

Traditional approaches to network management are inadequate to manage active networks. In section 2.2.1 we provide a concise overview of traditional approaches to network management. Subsequently, we identify the limitations and provide arguments for the adoption of the more dynamic approach that can be achieved via active networks in Section 2.2.2.

2.2.1 Traditional Approaches to Network Management

In this section we will briefly describe the most important approaches to traditional network management. Specifically, we consider TMN, TMF/NMF and TINA.

Telecommunications Management Network – The Telecommunications Management Network (TMN) supports management activities associated with telecommunication networks which consist of many types of analogue and digital telecommunication equipment and associated support equipment. TMN supports numerous operating systems for telecommunication management. Each operating system has management functions, which interact with potentially remote network elements or peer operating systems.

The impact of TMN within the industry has not fulfilled the initial expectations due to a number of factors. The availability of standards and the clear advantages of the object-oriented information model did not avoid delays in the implementation of TMN due both to technical and economical factors. Such factors included the inherent cost and complexity of the technology, the lack of application development environments, and the costs of substituting existing technologies with new, standard-based ones.

Tele-Management Forum – The TeleManagement Forum (TMF) and its member companies intended to identify, create, develop, and implement real world solutions that automate and streamline telecom operations. In other words, seek the most effective ways to improve public networks and services management.

The TMF activities are structured in programs. The major program areas include Process Automation Programs and Technology Integration Programs. Each of them is subdivided in teams or subprograms working on a specific field of the program. The TMF has always taken a pragmatic approach: real world and product solutions are sought. This entails the use of existing and mature technologies and products.

TINA – The Telecommunication Information Networking Architecture (TINA) Consortium was formed in 1993 with the intention of defining a common architecture upon which next generation telecommunication services could be built. The architecture proposed logically separated high level applications from the physical infrastructure. Central to the architecture was object-oriented distributed architectures such as the Object Management Groups and the support of Distributed Processing Environments (DPE). The architecture itself was defined through specifically identified components, clear separation points called Reference Points and guidelines on how to structure applications so that they could be easily integrated into the TINA architecture.

The TINA architecture offers certain components that can be applied directly for network management. Unfortunately, the full TINA architecture -–the network architecture in particular – is seen by some as being too heavyweight for most applications and services. As a result, it has often been the case that subsets of the overall TINA architecture have been implemented. Whilst the full TINA architecture was not completely accepted by the telecommunications and software industry more widely, many of the ideas that were contained within the TINA architecture were seen as useful, and as such are currently being re-used in other areas.

2.2.2 Why Traditional Network Management Approaches are Insufficient for Active Networks

During the last decades many 'traditional' -, non-active-, approaches for network management have been proposed with some approaches having gained more relevance in industry and/or through standardization organisations than others. However, none of these architectures is suitable to be used in active network management. Either because they cannot or are unable to, manage some of the new capabilities made possible by active networks.

Recent developments in the telecommunications arena have shown that customer requirements change rapidly with increasing demands on the infrastructure and services and protocols supported there. The process of standardisation, development, and deployment of new protocols is a laborious and drawn-out one. A representative example of this is the deployment of RSVP and IPv6, which has been widely recognised as useful; however, their network-wide deployment has still not taken place. As a result, many innovative applications that would require features of new protocols cannot currently be supported. Hence, a framework to support such network infrastructure dynamism is required.

Unfortunately, traditional network management approaches do not succeed in being able to cope with this required dynamism. These considerations lead us to the conclusion that dynamic network infrastructures are required expediting and facilitating the deployment and management of new network protocol software, thereby helping to solve the problems in today's networks. Active networks represent one viable way in which the dynamism of the network infrastructure can be supported directly. Active networks allow for example an active network management system to dynamically enhance its functionality to adapt to a new network condition or application.

2.2.3 Related Work on Active Networks and Management

This section serves as a concise overview on what has been done on the topic of active networks and management outside FAIN. We follow the logical division, made in Section 2.1.1, into 'active management of networks' in Section 2.2.3.1, and 'management of active networks' in Section 2.2.4.

2.2.3.1 Active Implementation of Traditional Management

Application of active technology to the management plane has been done by several research groups in the past. In this section an overview of these attempts is given. In general, network management applications aim at simplifying network management tasks, using AN-technology. Also, management tasks that were hard or even impossible to implement without AN-technology are realised. Most applications focus on *distributed management,* using AN techniques to delegate management tasks.

Table 1 gives an overview of known active networking management applications. Although not all applications are representative of the AN management approaches being developed in FAIN, they give a good overview of the current status of AN management activities

Table 1 – State of the Art on Active Implementations of the Management Plane.

Application	Owner	Remarks
Active distributed management, with several sample management applications	NTT network innovation lab/CTR Columbia Univ, Kawamura/Stadler [66]	New Internet capabilities ask for new management paradigm. Distributed control with AN technology. No further explanation
Distributed network management applications	Bell Labs/Lucent Raz/Shavitt [67]	A couple of examples: adaptive control, router configuration, element detection, network mapping, security management (intruder detection, fighting denial of services attacks) The idea is that AN can be used for fast and easy deployment of distributed network management. Distributed NM is needed to more efficiently use network resources in large and complex IP networks.
Distributed management, e.g. configuration determination	IBM Research, Feridun [68]	Use of active agents as management components. Again, distribution of management tasks.
Survivability of services, backup creation and location selection	Univ of Kansas, Minden & Evans [69]	How to survive malfunctions. Uses composable blocks.
Scriptable remote network management	Intel, Putzolu, Bakshi [70]	Phoenix framework: for easier network service deployment. This application makes use of mobile agents that can run monitoring scripts on remote locations (in stead of central location).
Topology discovery	Bell Labs, Raz & Shavitt [71]	The new algoritthm, report-on-tree, uses less bandwidth than Segall's report-direct PIF algorithm. AN is used to implement and execute this algorithm.
Software abstractions for network management applications	Univ. Pennsylvania & Bell Labs, Kornblum, Raz, Shavitt [72]	The ABLE platform offers abstractions for management applications, making it possible to abstract from heterogeneous native management interfaces. Improvements in flexibility and performance of network management have been shown. Demonstration of congestion avoidance used as a proof of concept for benefits of AN.
DARPA project on Active Virtual Network Management (AVNM)	GE CRD, Bush, Kulkarni [73]	GE developed active network algorithms that allow active, intermediate network nodes to predict their own behaviour and that make it possible for network equipment vendors to provide active logical processes that accomplish prediction for their devices. This innovation will make possible predictive network configuration and management.

2.2.3.2 Current Efforts in Management of Active Networks

In this section related work on management of active networks is described. Again, we do not claim to be exhaustive in this description, which contains both traditional and policy-based management styles.

Virtual Active Network – The concept of Virtual Active Networks (VAN) were proposed in in order to address the problem of service provisioning and management in a telecommunication environment that uses active networking technologies. VANs are perceived as a generalization of traditional Virtual Private Networks (VPNs). A provider sets up a VAN according to a stipulated contract with its customer. Since every customer gets their own VAN with virtual resources (bandwidth, processing power, memory, etc.), customers are isolated from each other.

A VAN can be described as a graph of virtual active network nodes (ANNs) interconnected by virtual links. In the Active Network Working Group, virtual active nodes are called Execution Environments (EEs). On a physical active node two different kinds of EEs can be installed: firstly, a privileged EE-type for the management of the physical ANN and secondly, a customer EE type. Initially, there will be only a management EE on a physical ANN. Solely one management EE exists on any ANN. The interconnection of these privileged EEs builds the management VAN. The management VAN spans over all physical active nodes of the provider's network. The management EE will create, modify, monitor and terminate customer EEs.

Abone – Currently, every node on the ABone runs active networking management software known as *Anetd*. It is an experimental software designed to support the deployment, operation and control of active networks. Specifically, *Anetd* allows for the:

- deployment, configuration, and control of networking software, including active networking execution environment (EE) prototypes;
- demultiplexing active network packets encapsulated using ANEP to multiple EEs located on the same network node and sharing the same input port.

Ponder – Ponder is a declarative, object-oriented language for specifying different types of policies, for grouping policies into roles and relationships, and then defining configurations of roles and relationships as management structures. Ponder can be used to specify security policies with role-based access control, as well as general-purpose management policies. It is intended to be extensible to cater for future types of policies.

In Ponder, a policy is a rule that can be used to change the behaviour of a system. Separating policies from the managers that interpret them allows the behaviour and strategy of the management system to be changed without re-coding the managers. The management system can then adapt to changing requirements by disabling policies or replacing old policies with new ones without shutting down the system.

2.2.4 Policy-Based Management of Active Networks

The use of policy-based style to the management of active networks is the subject of this section. Section 2.2.4.1 elaborates on the differences with traditional management styles, and Section 2.2.4.2 describes the advantages to be gained when using policy-based management of active networks. Finally, having identified this more powerful management technology, the challenges it has to face when managing active networks are described in Section 2.2.4.3.

2.2.4.1 Policy-based Management Style versus Traditional Management Style

The use of policies for network management has recently been introduced to the Internet community. However, for the deployment of Policy Based Network Management Systems in the Internet, a standardisation process is required, to ensure the interoperability between equipment from different vendors and PBNM systems from different developers. Both the Internet Engineering Task Force (IETF) and the Distributed Management Task Force (DMTF) are currently working on the definition of standards for Policy Based Network Management. The DMTF is mainly focused on the representation of policies and the specification of a corresponding information model and schema. The IETF is also working in that field, in co-operation with the DMTF, whilst also trying to define a general framework for a PBNM system, as well as a protocol that could be used for implementing a PBNM system.

The current goal of PBN is to provide facilities, which allow control of multiple types of devices that must work in concert across every single domain to provide a desired service. Examples of services, which can be controlled by PBN, are currently mainly Quality of Services (QoS) reservations. Examples of devices, which can be "PBN-enabled", are hosts (clients and servers), routers, switches, firewalls, bandwidth brokers, sublet bandwidth managers and network access servers.

Policies are defined as rules governing the allocation of network resources to certain applications/users. The IETF Policy WG defines a scalable framework for policy administration and distribution that will allow interoperability among the multiple devices that must work together to achieve a consistent implementation of a network administrator's policy. For this reason, directory schemas are standardised in order to enable the various network devices to interpret the configuration information consistently. For the distribution of policy information, IETF protocols such as LDAP, DIAMETER, COPS are used.

In comparison with previous traditional network management approaches, PBNM offers a more flexible, a and customisable management solution allowing each router/switch to be configured on the fly, for a specific application tailored for a customer. This flexibility, however, is not without cost. PBNMsystems have problems in security and scalability since they were originally thought to be used in a LAN-like network environment. Another problem in current PBNM is the semantics of policies. Actually, the policies that can be defined are limited by the current information model, which includes only classes for the representation of policy conditions based only on time and on packet headers. It is not possible to define new types of conditions, like conditions based on the status of the node or the network, without extending the core information model. Another problem of the current PBNM architecture is that it is only able to address fairly limited domain of issues, e.g. those that can be translated to fixed configuration settings. For example, QoS issues often needs complex interactions between relevant network management components. These complex interactions can not be easily implemented in the current PBNM architecture. Moreover, according to the policy framework, policies are defined or modified by an administrative tool and the intervention of the administrator is always required.

Active networks can resolve many of the problems inherent in current PBNM technology. They allow for enhancing the management architecture dynamically, for the introduction of new applications or device specific policies, tailored to realise complex tasks, and for the automation of e network management tasks. The Policy-Based Management architecture described in the next chapter shows how active networks can be used to overcome many management problems inherent in traditional management approaches.

2.2.4.2 Advantages of Policy-based Management Using Active Networks

In FAIN WP4, service logic for the PBN-based provisioning of a sample service (e.g. virtual private networks or QoS policies, depending on progress in the IETF) will be implemented on top of the Node API. With this case study, we expect to be able to demonstrate the following benefits of active networks:

- **flexibility**

Using active networks, policies can be expressed flexibly as active code as well as data structures - which are limited by the constraints imposed by standardised database schemas. Policies expressed, as active code will be able to have more inherent intelligence than traditional policies.

- **extensibility/dynamicity**
  Active networks technology will allow the PBNM system to dynamically extend its management functionality through downloading of new components. This will allow FAIN PBNM to be able to cope with changing network infrastructures or protocols.

- **automation and distribution of management tasks**
  One important goal of every management architecture is to automate and distribute tasks as much as possible. Active networks allow for the the automation and distribution of almost all management tasks, e.g. node self-reconfiguration due to changing network conditions.

- **delegation**
  The delegation of management tasks to customers is seen by network operators as an important property of a management architecture, since it will dramatically reduce their network management costs. Using active networks, a secure and effective way to delegate management tasks to customers who will gain more control over their contracted resources while network operators will save funds.

- **applications use of management**
  The ideal network and node configuration for each application might be different, and might change depending on network conditions. Active networks provides the technology to allow applications to customize their assigned resources making use of the management system, i.e., through the use of policies way, they can achieve an optimum usage of the application assigned resources in different network or node conditions in a secure manner.

- **customer-specific management**
  Many reasons can lead a customer to desire the use of their management system to configure, monitor and control their assigned resources, e.g. where deep knowledge of the possibilities specific to their needs is available, where existing, high level management applications already available for that system etc. Active networks technology allows customers to use their own management system to configure node and network resources in a secure way.

- **simplified service deployment**
  Since all the FAIN nodes support the same Node API, we expect that the same PBNM service logic may run on the different nodes. In contrast to current approaches, where the service logic has to be developed individually by each node vendor, this results in an improvement on the service development and deployment process with respect to costs and time to market.

- **benchmarking**
  Since PBNM-enabled network devices are expected to be available on the market soon, the PBNM case study allows benchmarking with existing PBN implementations, in particular with respect to performance, interoperability and reliability.

2.2.4.3 Challenges in the Management of Active Networks

The main question for the PBANM system to address is:

'in what respect is PBNM of active nodes different from PBNM of traditional nodes?'

The answer should first deal with the comparison between passive/traditional management of active and passive nodes irrespective of whether it is policy-based or not.

Element-level administration usually handles configuration and fault management manually, using generic remote access capabilities (e.g., telnet in Unix), rather than a specialised protocol although some management applications do use SNMP to access and manipulate settings. However, configuration management (CM) functions are often too complex to handle via SNMP and are thus accomplished by scripts at the element under the control of the network management station (NMS).

The use of this unstructured and ad-hoc manual approach to traditional element management is a significant barrier to the efficient, secure and robust operation of active networks. As such active networks need to be managed differently. Consider using SNMP-based management to manage an active network. Some requirements are envisaged:

- when an active element is loaded into a node, it is necessary to load respective instrumentation and MIB components and integrate these with the element management software. It is also necessary to load similar MIB components into the NMS to enable management application tools to access the enhanced feature on the active node;
- these dynamic changes at element management level and the network management level will have to be synchronized and coordinated with the dynamic changes of the active network;
- MIB structures at element level and network level will have to change dynamically on a time scale similar to the time over which MIB variables change. Furthermore, MIB data may have to persist even if the respective active application has terminated, to facilitate analysis of potential problems or recovery of configuration states.

In contrast with traditional network applications, which are entirely separated from management software, active applications need to integrate monitoring and control capabilities. The traditional approach to network software design has been to incorporate function-specific monitoring and control capabilities with every protocol/network-system. For example, routing software uses the Routing Information Protocol (RIP) to monitor network topology information, and transport-layer software can use RSVP to control allocation of resources. Each such application requires its own specialized instrumentation and access protocol to facilitate monitoring and control functions.

Thus, in contrast with current networks, active networks require an integration of management mechanisms and application software. Clearly, the management framework offered by SNMP is unable to handle the management needs of active networks.

2.2.5 Approach taken within FAIN

2.2.5.1 FAIN and the TeleManagement Forum

This section describes the top-down conceptual approach of the management system for FAIN guided by the ideas and principles of the TeleManagement Forum (TMF). The TMF has not only defined the TMN-model on which FAIN has defined its focus. It also has defined a detailed map of all functions that must be present on each level. The Telecom Operations Map (TOM) describes blocks of functions that must be implemented by either an automated or a human process. These TOM function blocks will be used to ensure the completeness of our architecture. In Appendix A a more elaborate introduction into the TMN is included. A simplified representation of the TMN-model is given below.

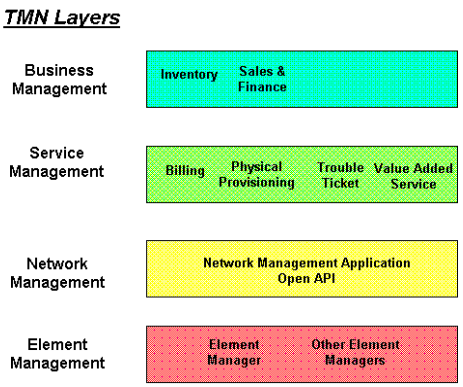

Figure 5 – The layers of the TMN-model and some of the functionalities at each level

In the top-down conceptual approach three steps can be identified. The first is to define which TOM-functionalities are important for us. Initially, the minimum set of requirements prescribed by TOM will be taken as the functionalities. The second step is to design a functional architecture based on these TOM-functionalities. The third step is the technical implementation of the functional architecture. In the subsequent chapters these high-level ideas will be elaborated.

Notice that only at the third step a selection of an implementation technology is made. The implementation can be done using policy-based techniques or by means of active implementation of traditional management techniques. In practice we expect both implementations, as described in Section 2.2.3.1 and Section 2.2.4, will be pursued within FAIN depending on availability/maturity of the technology, and suitability for implementing a particular function.

As explained in Section 2.1.1, FAIN focusses on the two lower levels of the TMN-model: the Network Management layer and the Element Manager layer. In this section the functionalities for the network management layer are described. Also an initial architecture in which these requirements are met will be presented. The minimal set of functions for the Network Management layer, as prescribed by TOM are:

- **network planning and development**
  Planning of network capacity and capacity migration. Also plans the logical network configuration and physical sites.
- **network provisioning**
  (re-)Configures the network and administrates the logical network (includes setting identifiers for services). Also tests the network to ensure operational readiness
- **network inventory management**
  Installs and administers the physical network. Repairs and does maintenance to the network. Aligns the inventory data with the actual network.
- **network maintenance and restoration**
  Does problem analyses and testing. Maintains network quality and keeps a history of network problems, tests and maintenance tasks.
- **network data management**
  Does collection, correlation and formatting of usage data and events. Gives notifications when the performance of the network degrades and does traffic control.

2.2.5.2 Active Networks have Impact on the Management Functions

The initial FAIN management architecture should reflect the requirements that the TOM-functions described above. These requirements can be fulfilled by traditional network management or by active management. As mentioned earlier in Section 2.1.1 it is expected that active management has advantages over traditional management. The term 'active' means that (more) intelligence is implemented at lower TMF-levels. With this intelligence (management) decisions can be executed lower in the network, which is more efficient and flexible because fewer layers are involved in the process. The extra capabilities the active networks offer for management purposes are *aggregation* or *pre-processing* and *consciousness*. The impact of these added capabilities for management purposes is described in the remainder of this section.

Aggregation – The network elements are able to aggregate network data and events to enriched management information. This information is passed to the network management layer instead of the entire flow of network management data and events. The Network Management layer is then able to react more efficiently, flexiblye and intelligently using this enriched information. This would mean that the Active Node elements are able to collect, correlate and format management data and events in a distributed manner.

Consciousness – The network elements are able to take conscious management decisions(i.e. based on network status) at element level. This consciousness of the network means that the Active Node element is intelligent enough to identify performance degradation, capacity problems and physical problems and take appropriate action itself. This would mean a network that would react to situations without the involvement of the Network Management layer at all.

When looking at these two manifestations of activeness in the network it is clear that the traditional TMF separation between Network Element and Network Management layer is blurred. Some functionalities traditionally carried out at the network management level now drop to the element level. The network element level is not the focus of the TOM-model and thus TOM has no functions or requirement for this level. For every function block of TOM activeness has its effects:

- **Network planning and development**
  Network elements could be given performance goals with rules what to do when these are not met. For example change their routing table. Planning no longer only concerns bandwidth capacity but also processor power and storage capacity. The active nodes could do part of the logical network configuration.

- **Network provisioning**

The active nodes could partly configure itself or negoatiate a configuration with its neighbours. It could also test operational readiness itself.

- **network inventory management**
  The inventory management of the actual network could be automatically maintained by the active nodes.

- **network maintenance and restoration**
  An active node which encounters a problem could try different actions/configurations to solve or circumvent these.

- **network data management**
  Active nodes could collect, partly correlate and format usage data and events before sending these to the Management level. It could also notify the Network Management layer when performance degrades below a certain level, or take action itself.

From the above examples, it becomes clear that using active networks technology for management purposes does change the way management has traditionally been implemented. In all cases, a network element processes network management data itself and in some cases it doesn't even consult the Network Management level before taking action. It is therefore expected that this change results in advantages with respect to scalability and flexibility of the management system.

2.2.6 FAIN Methodology

In order to cover both element and network level management aspects, we initially adopt the same architecture for both levels. Our initial architecture is essentially the policy-based networking framework adopted by the IETF, enabling us to control active nodes and active networks using the same framework. We elaborate on both this initial framework and on the applicability of this framework at both element and network level in later chapters. Since different functions are required at each level, we might expect some differences in the capabilities required at the two levels. Thus it is expected that in further iterations of the management system, the architecture for element level will be modified with element-specific functionalities, while the architecture for the network management layer will be modified with network-specific functionalities. In other words, the initially identical architectures are expected to diverge during future iterations.

Anticipating future research, we pursue architecture with an explicit separation between element and network levels. Also we study the relations, and in particularly the transformations (or mappings) between functions/policies at both levels. Though no comprehensive research will be conducted within FAIN on the service management layer, we will look at interfacing to the service management layer.

2.3 SYSTEM REQUIREMENTS ON THE FAIN MANAGEMENT ARCHITECTURE

To derive the specific system requirements upon which the management architecture will be developed and subsequently implemented, we consider generic requirements likely to be associated with the Network Management System (NMS) and the Network Element Management System.

With regard to the NMS, there are several key aspects, which should be considered during requirement analysis. Specifically we consider system testing purposes and functional aspects of the NMS.

2.3.1 Requirements on the NMS for Testing Purposes

Associated Service Level Agreements (SLA) should allow delegation of management functionality. This delegation can be specified up to a certain depth level,e.g. the Network Infrastructure Provider (NIP) (see Deliverable D1 [103]) delegates to the Network Service Provider (NSP) (see Deliverable D1 [103]), and the NSP can delegate functionality to the Service Provider (SP) as well, but the SP can not delegate management functionality to the customer.

- The SLA should determine which concrete management functionality is going to be delegated.
- The SLA should establish a determined level of Quality of Service of bandwidth and forwarding resources.
- The SLA should be able to determine a certain level of security in the service offered.
- In SLA negotiation the management system (in representation of the NIP) should detail the price of each of the offered resources.
- The management system should support re-negotiation of a SLA.
- The management system should support inter-domain management.
- The management system has to support policies that provide management functionality of general offered services.
- The management system has to support metapolicies, which allow the delegation of this functionality.

- The SLA should be able to determine atomic setting of offered resources.

2.3.2 Functional Requirements On the NMS

These requirements have been classified into the three main areas for network level management specified in the Telecom Operations Map . Specifically we consider network provisioning, maintenance and restoration and data management

2.3.3 Network Provisioning Requirements on the NMS

- The NMS must support active policies for network connectivity management.
- The NMS must support metapolicies for delegation of computing and forwarding resources of the management system to a user-specific management system (see 2.4.1.3.2.5).
- The NMS must support metapolicies for control of the policy lifecycle.
- The management system at the network level should be able to support atomic sets of policies. It should check whether all policies can be enforced and, otherwise, remove them at all.
- The NMS must support metapolicies that allow delegation of management functionality at the network level.
- The management system should be able to check if a user is allowed to use management functionality, to delegate it and if so up to which level.
- The NMS must support policies with explicit QoS requirements.
- The NMS must select the network resources according to the required QoS
- The NMS must receive notifications from the Element Management Systems (EMS)s about the degree of enforcement of the policies sent to them
- The NMS must keep track of all the established connectivity
- The NMS must provide facilities for graphical visualisation of established connectivity services (optional)
- The NMS must support the protocols and network technologies needed to establish offered services
- The NMS must be able to modify already established connectivity services
- The NMS must be able to terminate network connectivity services releasing all the involved resources
- The NMS must be able to send appropriate request to the EMS to support the modification or removal of network connectivity services
- The NMS must be notified about the addition and/or deletion of network resources

2.3.4 Network Maintenance and Restoration Requirements on the NMS

- The NMS must receive and process state reports from the EMSs
- The NMS must inform the appropriate user about current operational status of the network managed resources and if necessary propagate these reports upwards to client layer through the appropriated interfaces
- The NMS must be able to apply alarm filtering to avoid superfluous information according to specific user-reconfigurable policies

- Alarm filtering must be applied either in regards to the user and to the management client layer
- The NMS must be able to receive alarm filtering criteria from the client layer
- The NMS must be able to install appropriate filters in the EMSs
- The NMS must be able to determine the root cause alarm suppressing all the other correlated alarms.
- The NMS must identify the affected resources in cause of failure at the appropriate granularity level
- The NMS must provide facilities to visualize and manipulate the existing system alarms
- The user must be able to acknowledge and clear alarms
- The NMS must provide a mechanism for synchronisation of alarm information
- The NMS must provide a network recovery mechanism in case of failure
- If legacy (passive) nodes and active nodes are co-existing, when an active node is down, the management system should find out the appropriate active node, in order to migrate the active circumstances.
- When an active node is down, the management system should assign another active node which has sufficient and capable resources. This active node might be chosen based on its proximity to the failed node.
- When an active node is down and an alternate active node could not be found in its managed domain, the management system might negotiate with other managed domains to assign an active node for replacement.
- After replacement of an active node or partial migration of resources, the management system should check whether existing SLAs can be maintained.

2.3.5 Network Data Management Requirements on the NMS

- The NMS must support policies for monitoring purposes.
- The management system should send monitoring reports to a user as a result of a monitoring policy enforcement by that user.
- The NMS must monitor the QoS of the network connectivity services supported.
- The NMS must support the setting of filtering for performance reporting
- The NMS must support the start and the stop of performance monitoring reports
- The NMS must be able to store performance data in performance logs
- The NMS must support facilities for performance data visualization and browsing
- The NMS must support the setting of performance thresholds to generate performance notifications
- The NMS must be able to receive performance notifications from the EMSs

2.3.6 Network Provisioning Requirements on EMS

- Support of metapolicies allowing delegation of management functionality at the element level.

- Support policies for the configuration of the Resource Control Framework (see Deliverable D2 [103]) component of an active node.
- Support policies for the configuration of Virtual Environments (see Deliverable D2 [103]) installed in an active node (optional).
- The element level should be able to provide to the user a pointer to the interface where user's active code should be installed in an active node in order to take advantage of the resources already reserved on that code.
- The management system should be able to treat requests from the Active Service Provisioning system asking whether a certain user is allowed to install code in an active node.
- Support of policies for the allocation of traffic flows to EEs (see Deliverable D2 [XXX]).
- The management system should be able to check if a user is allowed to use management functionality, to delegate it and if so up to which level.
- The management system should allow user-specific policies.
- The EMS must support metapolicies for the delegation of computing and forwarding resources of the management system to a user-specific management system.
- The EMS must support metapolicies that control the policy lifecycle.

2.3.7 Network Maintenance and Restoration Requirements on EMS

- The EMS must be able to send state reports to the NMS
- The EMS must inform the appropriate user and/or the NMS about current operational status of the active node managed resources
- The EMS must be able to apply alarm filtering to avoid superfluous information according to specific user-reconfigurable policies
- Alarm filtering must be applied either in regards to the user and to the NMS
- The EMS must be able to receive alarm filtering criteria from the NMS
- The EMS must be able to determine the root cause alarm suppressing all the other correlated alarms.
- The EMS must identify the affected resources in cause of failure at the appropriate granularity level
- The EMS must provide facilities to visualize and manipulate the existing active node alarms
- The user must be able to acknowledge and clear alarms
- The EMS must provide a mechanism for synchronisation of alarm information
- The EMS must provide a node resources recovery mechanisms in case of failure
- Repair and replacement should not be done only for every element or node, but for every resource. For instance, when a certain storage facility is faulty, standby storage should be assigned. This standby storage could be in the same node or in other node.
- Recovery of fault in network element should have some interrelation with Active Service Provisioning (ASP). In other words, for example, the fault management system should might send notify alarm notifications that identify missing types of active codes to the ASP module. These active codes might subsequently be reloaded in alternative active nodes or into the repaired node.

2.3.7.1.1 Network Data Management Requirements on EMS

- The EMS must support policies that provide monitoring functionality.
- The management system should send monitoring reports to a user as a result of a monitoring policy enforcement by that user.
- The EMS must support the setting of filtering for performance reporting
- The EMS must support the start and the stop of performance monitoring reports
- The EMS must be able to store performance data in performance logs
- The EMS must support facilities for performance data visualization and browsing

The EMS must support the setting of performance thresholds to generate performance notifications

2.4 R12 - INITIAL FAIN ARCHITECTURE

In this chapter a description of the initial Policy-based Management architecture approach taken in FAIN is given. The architecture described copes with the management requirements exposed in chapter 2.3. However, the architecture described is general in the sense that it is management level independent. Therefore, the concrete issues of the architecture at the management levels covered by FAIN are described in chapter 2.5. Finally, the proposed implementation and the testing of the architecture designed is described in chapter 2.6.

The description of the architecture is decomposed in three main blocks:

- A high level definition of the architecture and functionality (section 2.4.1)
- A detailed description of the functionality and components of the architecture (section 2.4.2),
- And a detailed description of the interfaces between the components of the architecture (section 2.4.3).

2.4.1 Policy-Based Architecture Design

This section provides an introductory approach to the Policy-based Management architecture specified throughout the chapter. This section is further decomposed in four sub-sections. In section 2.4.1.1 the main characteristics of the architecture designed are specified. Section 2.4.1.2 describes how the main actors that interact with the management architecture are mapped to the FAIN Enterprise model. In section 2.4.1.3 the architecture is introduced through a static diagram where the main components of the architecture are depicted, and some dynamic diagrams that describe how the main characteristics of the architecture detailed in section 2.4.1.1 are supported by the architecture. Finally, section 2.4.1.4 describes the relation of the FAIN Policy-based management architecture with the IETF policy framework and other IETF policy standards.

2.4.1.1 Introduction

Active Networks have of a double impact on network management systems. That is, they provide to the management systems new tools to manage the network, i.e. larger configurability of node resources, code mobility, etc., while, on the other hand, they are more difficult to manage since new management requirements appear.

The Network Management System described in this document tries to cope with this environment, making use of the advantages of active networks.

The proposed PBM system is based on Policy Technology in order to take advantage of the scalability and interoperability properties of policies. Therefore, the backbone of the architecture is made of a Policy Decision Point (PDP), Policy Enforcement Point (PEP), a policy conflict check component, and the local policy database. The local policy database will be the repository of all policies, currently enforced or enforceable when the conditions are met in the managed active network.

In order to fulfil the requirements related with active networks stated above other specific components have been identified. Since different users (or applications on their behalf), can set policies in the system a credential check component and a meta-policy database component has been introduced in the system. The policy conflict checking is done inside each PDP. To cope with the self-configuration requirement the PDP functionality has been extended. The management functionality extensibility is achieved through the dynamic downloading of new PDP-PEPs to the PBM system. The extended PDP is decomposed in several sub-components which will be described in further detail at section 2.4.

2.4.1.2 Mapping of the management system to the FAIN Enterprise model

In this chapter a brief overview of the FAIN Enterprise model is briefly summarised. The mapping between management systems and actors, and FAIN business roles is given as well. Finally, management mechanisms required at the corresponding Reference Points are discussed.

2.4.1.2.1 Overview of the FAIN Business Roles and Reference Points

The FAIN enterprise model is depicted in the figure below. It contains four groups of business roles:

- service and network users (“Consumer”).
- Service and network providers (“Retailer/Service Provider”, “Active Network Service Provider” and “Network Infrastructure Provider”).
- software and hardware component manufacturers (“Service Component Manufacturer”, “Active Middleware Manufacturer” and “Hardware Manufacturer”).
- software component distribution mechanisms, i.e. a kind of service component “yellow pages” (“Broker”).

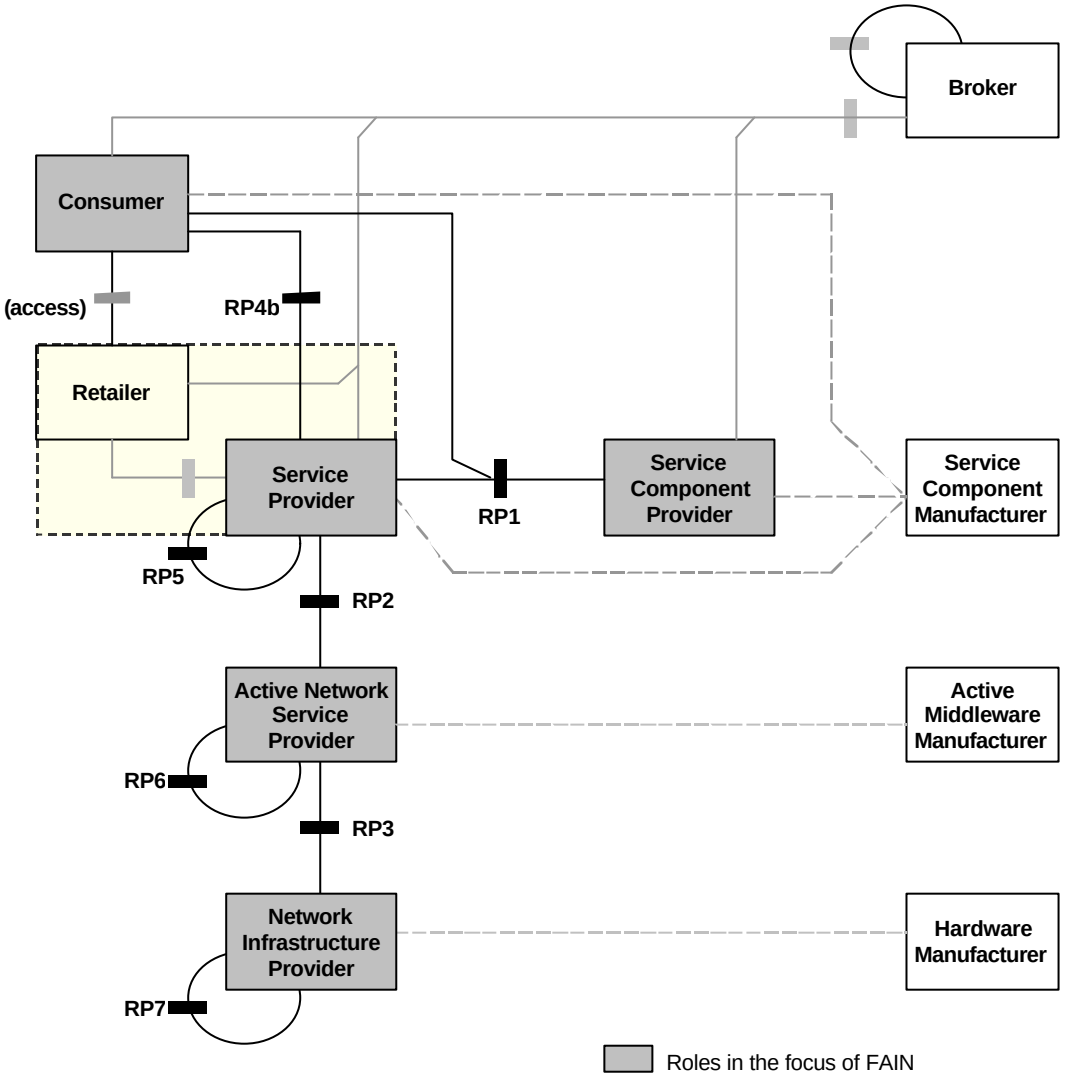

Figure 6 - FAIN Enterprise model

The business roles as understood here are strictly involved in the active networking business, i.e. they reflect the additional activities/properties, which are necessary solely to facilitate active networking. They, however, share some commonalties (common activities and properties) with their non-active counterparts (i.e. the providers of non-active services, connectivity providers, etc.). We see these roles mostly in the light of the TINA business model. The respective differences will be discussed later in the document.

For a detail description of the various RPs, please refer to the FAIN deliverable D1 [XXX].

2.4.1.2.2 Mapping of NMS Actors with FAIN BM Actors

The Network Management System (NMS) developed within FAIN can be decomposed in two systems, one at the network level (PBANM), and one at the element level (PBANEM). Both levels will interact with one another and with other actors, i.e. the users, the active nodes (ANN) and the Active Service Provisioning system (ASP).

In the figure below we show how all these management related actors are mapped with the FAIN Enterprise model actors. Both levels of the Network Management System (i.e. PBANEM and PBANM), the ANN and the ASP (at least the part of it that interacts with the management system) are owned by the Network Infrastructure Provider actor of the Enterprise model. Also, the FAIN Enterprise model actors that might, potentially, be able to interact with the management system are the Customer, the Retailer/Service Provider and theActive Network Service Provider. Therefore, they are mapped with the User management related actor.

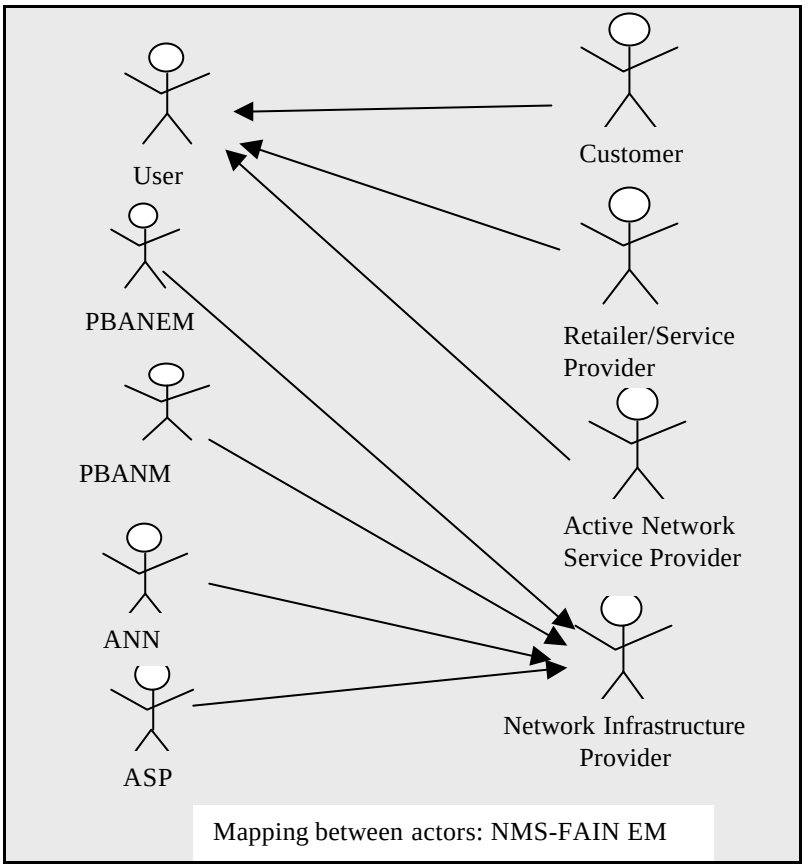

Figure 7 - Mapping Between Actors

2.4.1.2.3 Mapping of FAIN Bussiness Model RPs to NMS functions

From the previous section we get two main conclusions relevant to the mapping of NMS to FAIN Enterprise models RPs:

· The Network Infrastructure Provider owns, and is responsible for the PBANM, PBANEM, ANN and ASP systems.

· The Customer, the Retailer/Service Provider and theActive Network Service Provider, are mapped to single users who are able to interact with the management system.

The first conclusion implies that all relations between the PBANM, PBANEM, ANN and ASP systems, are NIP internal interactions, and therefore are not related with any Reference Point.

We can draw the second conclusion from the fact that the Customer, the Retailer/Service Provider and theActive Network Service Provider can make use of network management functions only if they have them delegated by the NIP. Therefore, when they make use of the management functionalities they are acting as the same actor.

These two conclusions brings us to the fact that the only reference points where network management mechanisms can be potentially used are the RP3 and RP7. However, the inter-domain RP7 is not consider in the following discussion, because it is not discussed in details in the FAIN enterprise model.

2.4.1.2.3.1 Management Mechanisms Associated With RP3

Below management mechanisms for RP3 are given:

- · A ANSP can negotiate a SLA with the NIP.
- The ANSP can request a re-negotiation of a SLA.
- The management system can report to the user-detailed information about the price of the offered resources.
- The management system can periodically report to the user on their assigned resources consumption and status.
- The user can:
  - Use the management system to configure their assigned resources.
  - Requests atomic sets of management actions, of which either all or none are done.
  - Use the management system to monitor and control a service.
  - Use the management system to delegate management mechanisms to one of their customers.
  - Use the management system to request the installation of active code in their assigned resources.
  - Use the management system to request the ability of installing active code by themselves at any time on its assigned resources.
  - Use the management system to request the ability of managing their assigned resources with their own management system.
  - Request to the management system that their assigned resources are isolated from others assigned resources.

2.4.1.3 Policy-based Management Architecture Overview

2.4.1.3.1 Static Diagram

In the figure below the Policy-based management architecture approach taken in the project is depicted and the main components of the proposed architecture are detailed.

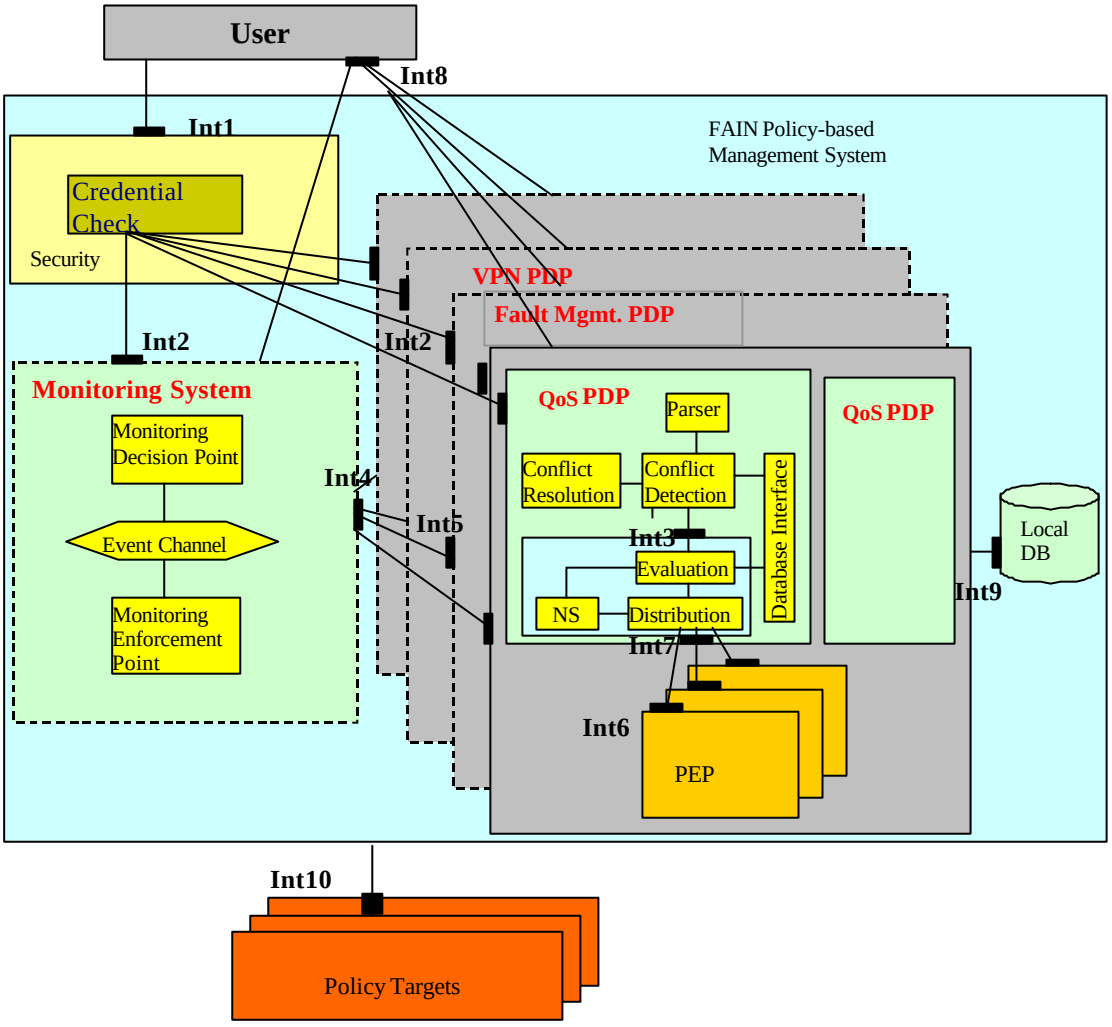

Figure 8 - PBM Architecture

2.4.1.3.2 Network Management System Scenarios

In this chapter we aim to provide a set of representative scenarios in order to show, and better understand, the behaviour of the presented architecture. The chosen scenarios, in our opinion, represent the functionality by which the architecture fulfils the basic requirements given previously set to the architecture. These scenarios are:

- Network Provisioning Scenario
- Signalling Scenario
- Reconfiguration Due to Network Status Scenario
- Application Provisioning Scenario
- User-specific policies Scenario
- Alarm Event Scenario

We have to note that although in the sequence diagrams of the scenarios, the Policy Conflict Check is shown as a different box of the PDP, this is made just to remark the policy conflict checking functionality as a key functionality in Policy-based Management. In the architecture, the policy conflict checking components are within each PDP.

2.4.1.3.2.1 Network Provisioning Scenario

This scenario shows the normal behaviour of the policy based management system, where a set of provisioning policies are received to be processed.

Pre-condition: The correct policy allowing the principal to set policies should be previously installed.

The interaction between components in this scenario will be:

1.- A set of policies are received in the Network Management System (i.e. in the correspondent PDP) along with a credential of the principal that sends them.

2.- The security checks component receives the policies and checks if the actor with that credential is allowed to set policies, using the metapolicies database.

2.1.- If the credential its incorrect then an error message is sent.

2.2.- If the credential is correct the policies are passed to the policy conflict check sub-component.

3.- The policy conflict check sub-component determines the consistency of the incoming policy with policies already set by that PDP. Possible conflicts are tried to be solve at this step also. Metapolicies can also be used to ease this process.

3.1.- If an unresolvable conflict is found an error message is sent back to the principal who sent the policy and the new policy is rejected.

3.2.- In any other case the process continues to the following step.

4.- The policy conflict check passes the policies to the PDP that looks for the PEP that will be in charge of each policy.

4.1.- If no appropriate PEP is found, then the PEP manager might ask to the Active Service Provisioning (ASP) system to download the PEP.

4.1.1.- If the PEP could not be downloaded then an error message is sent back to the network manager.

4.1.2.- In any other case the process goes ahead.

4.2.- The PDP should then install the downloaded PEP within the policy based management architecture.

5.- The PDP will then register in the monitoring system the events it wants to receive in order to be able to decide when those policies should be applied, and stores the policies in the database.

6.- The monitoring system sends, when appropriate, the registered events to the PDP.

7.- The PDP processes the events and decides when the policy should be applied.

8.- The PDP then retrieves the policies from the database and passes them to the correspondent PEP.

9.- Upon the reception of the policies the PEP realises the appropriate actions.

10.- The PEP informs the PDP of the result of the enforcement actions.

11.- Finally, the PDP informs the principal and the network level (only the element level, obviously) of any configuration change in the resources of the node due to the successful enforcement of the policy.

2.4.1.3.2.1.1 Sequence Diagram – Main Branch: PEP Previously Installed

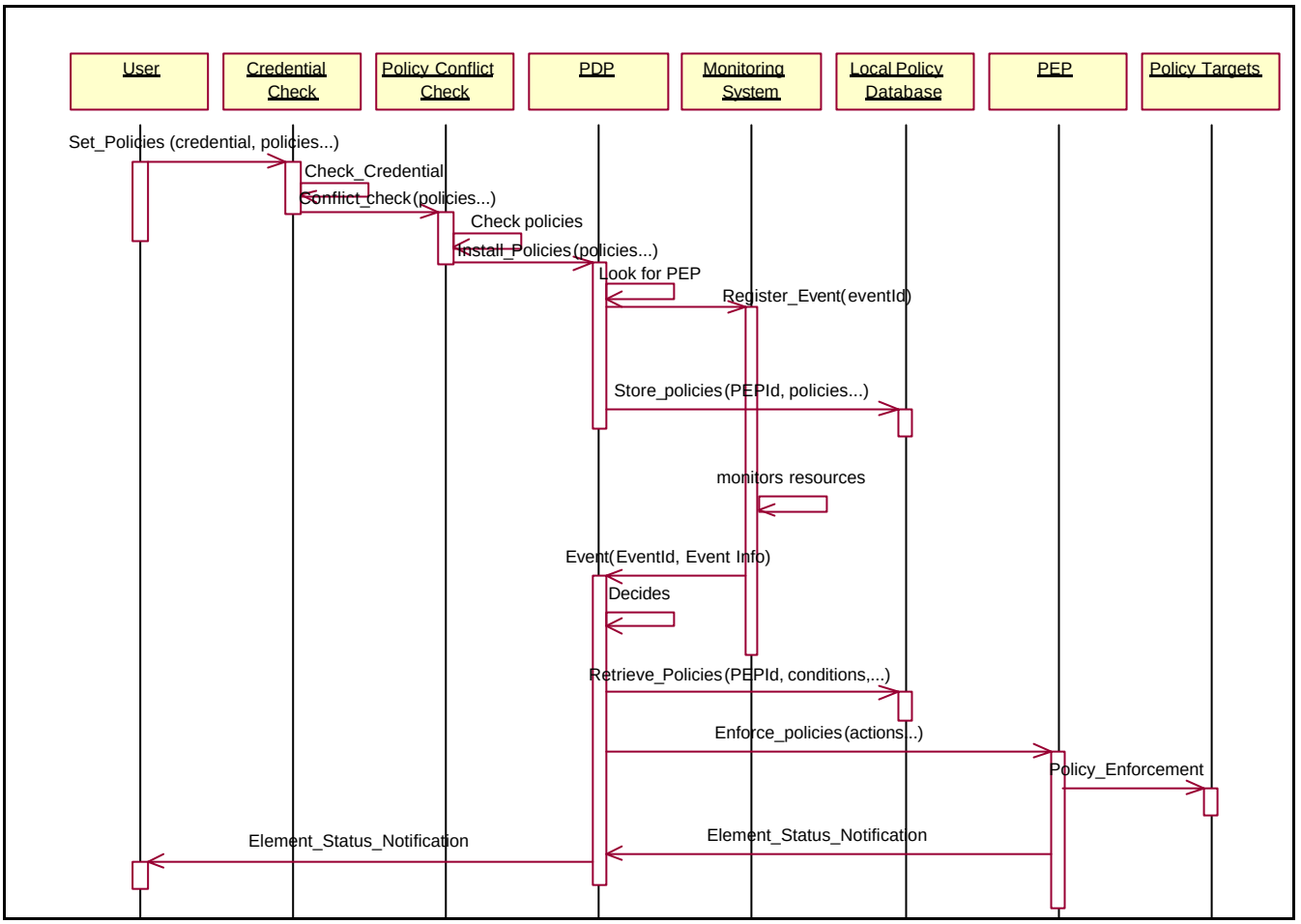

Figure 9 - Network Provisioning Scenario Sequence Diagram – Main branch

2.4.1.3.2.1.2 Sequence Diagram – Sub-branch: PEP Not Installed

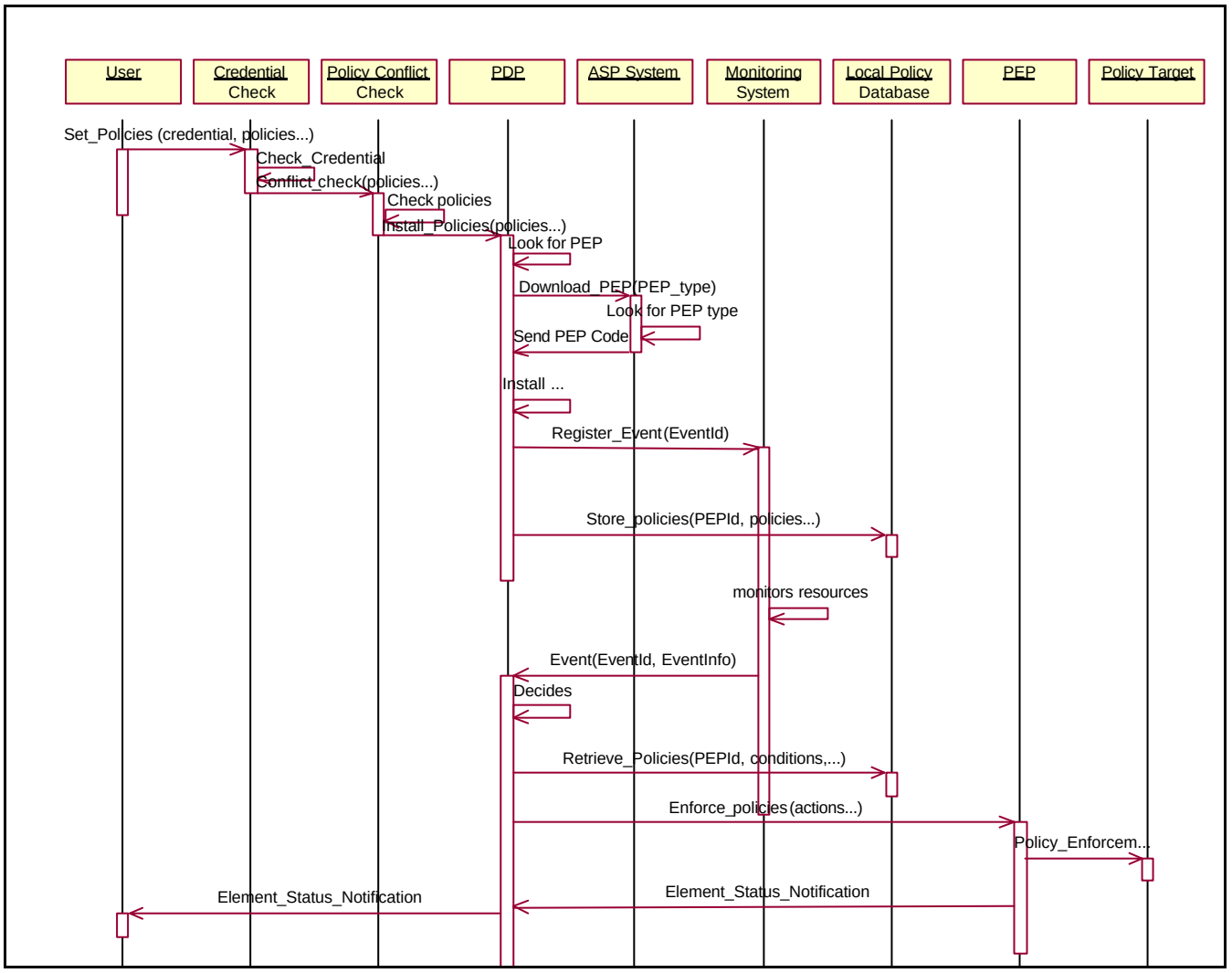

Figure 10 - Network Provisioning Scenario Sequence Diagram – Sub-branch

2.4.1.3.2.1.3 Activity Diagram

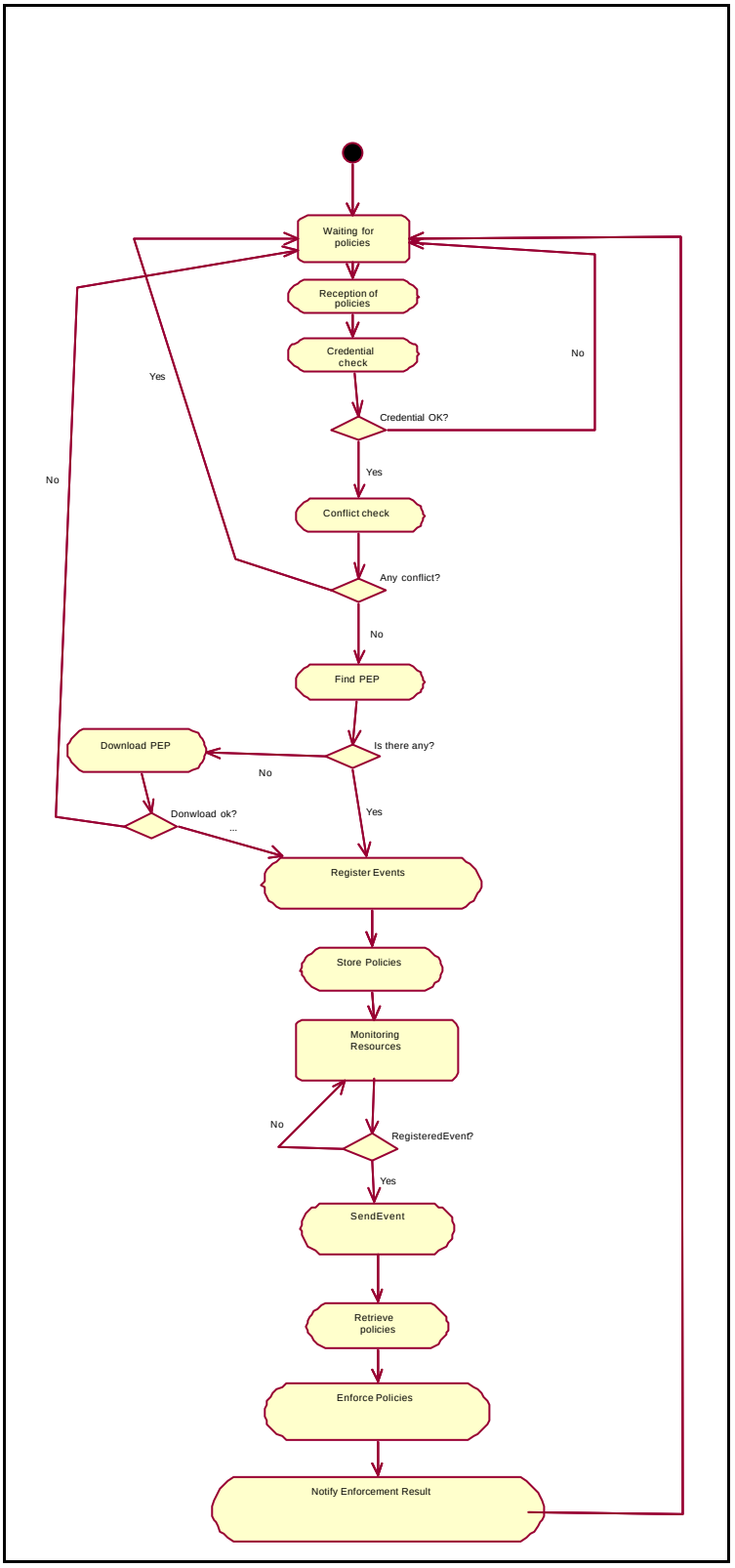

Figure 11 - Network Provisioning Scenario Activity Diagram

2.4.1.3.2.2 Signalling Scenario[1]

The goal of this scenario is to show how the node can reserve resources, say bandwidth, requested by a signalling packet (e.g. an RSVP packet).

1.- The signalling-reservation capable policy target sends the query up to the appropriate PEP along with a credential of the actor who wants to reserve the resources[2].

2.- The PEP asks the PDP for a decision.

3.- The PDP will check if there is any policy in the database allowing that customer to reserve node resources, and if so how many.

3.1.- If the customer is not allowed to reserve node resources an error message is sent back to the PEP that sent the query.

4.- The PDP will then ask the monitoring system for the resources already assigned to that customer.

4.1.- If the requested resources plus the resources already assigned are higher than the maximum resources allowed an error message is sent back to the PEP.

5.- The PDP tells the PEP to realise the corresponding reservation actions.

6.- Finally, the PDP informs the network level of the actions taken in the resources of the node due to the successful processing of the request.

[1] This scenario is only applicable to the element level instance of the architecture.

[2] We may consider also the possibility of having one PEP that receives unknown queries and that asks the PEP manager for a PEP that will be able to process them.

2.4.1.3.2.2.1 Sequence Diagram

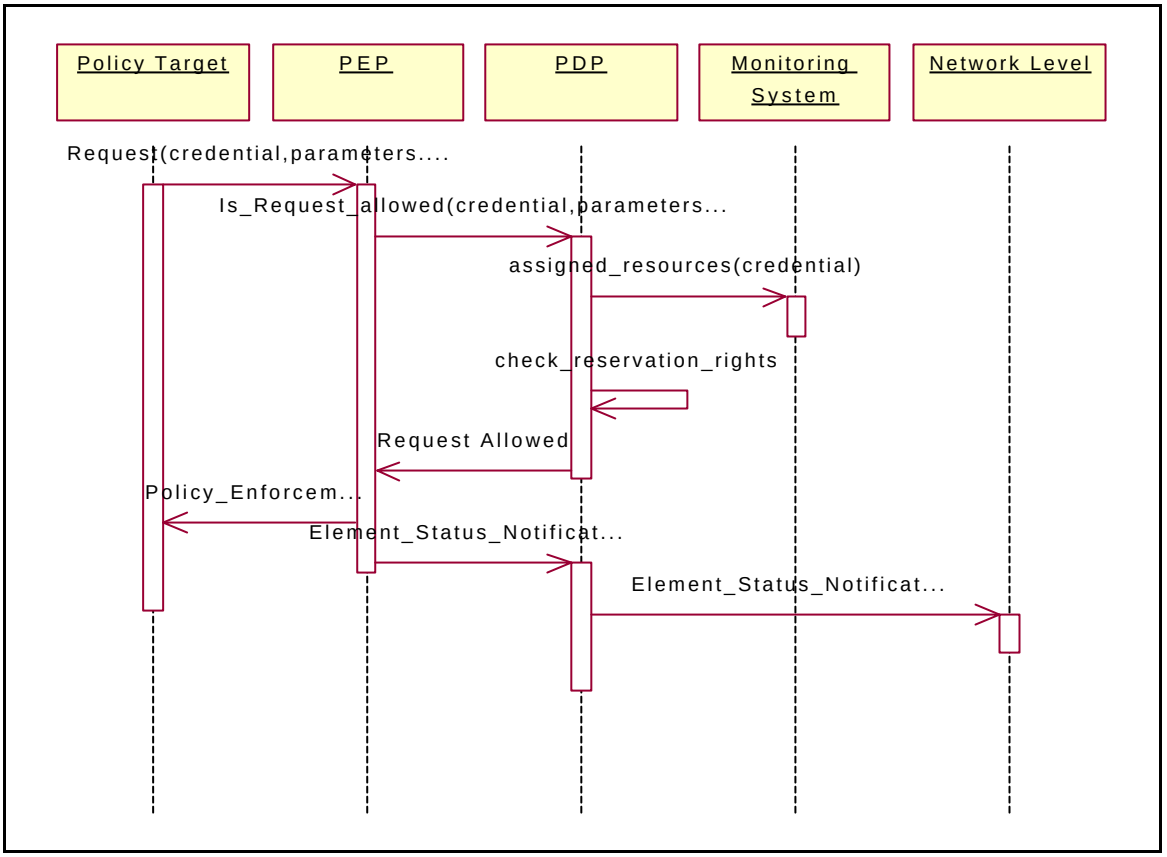

Figure 12 - Signalling Scenario Sequence Diagram

2.4.1.3.2.2.2 Activity Diagram

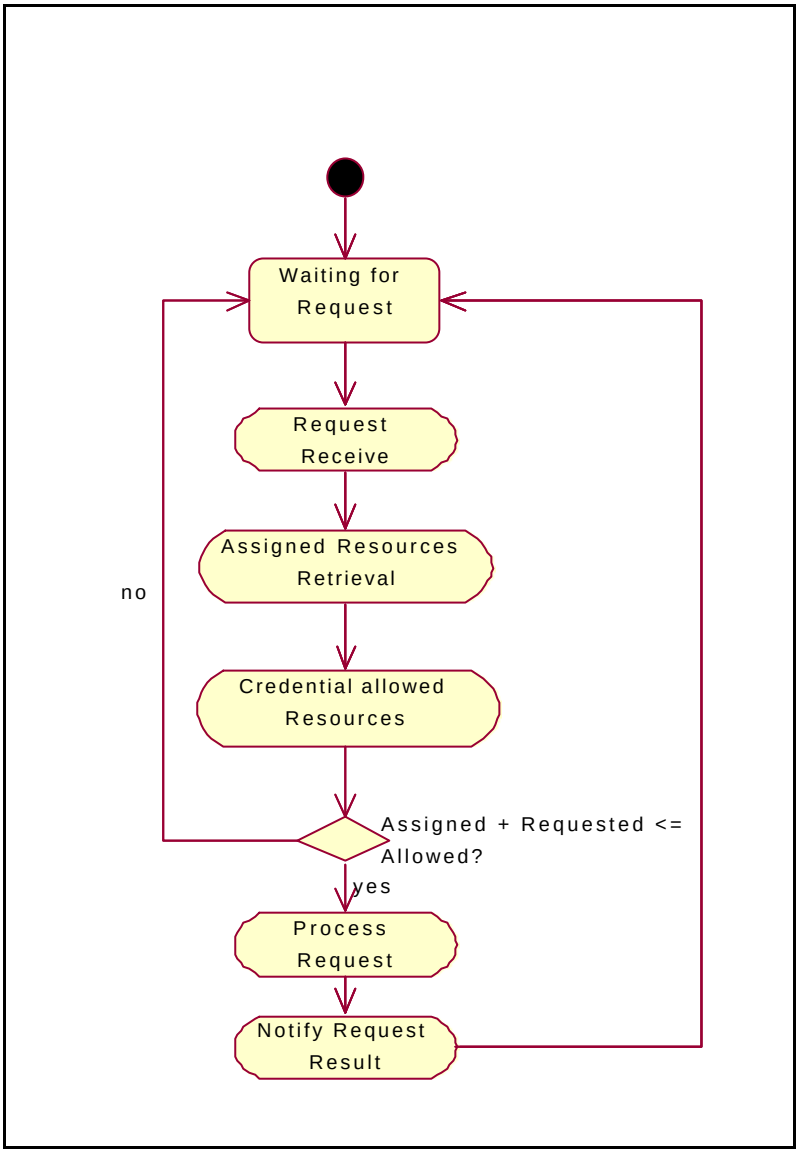

Figure 13 - Signalling Scenario Activity Diagram

2.4.1.3.2.3 Reconfiguration Due to Network Status

This scenario shows how the configuration of the active network adapts to changing network status.

Pre-condition: Sets of policies with conditions related to the network status should have been previously installed in the policy database.

1.- The monitoring system detects a change in one, or more, of the properties that reflects network status. It then checks if any PDP has registered events related with the status of that property.

1.1.- If there is no event related to that property, it does nothing.

1.2.- In any other case, it sends an event to the PDP that has registered it.

2.- The PDP will then fetch the policy or policies that satisfy the new network conditions from the policy database and will send these policies to the appropriate PEPs.

3.- The PEPs will realise the appropriate actions on the policy targets.

3.- Finally, the PDP informs the network level (only if it is at the element level) of any configuration change in the resources of the node due to the successful enforcement of the policy.

2.4.1.3.2.3.1 Sequence Diagram

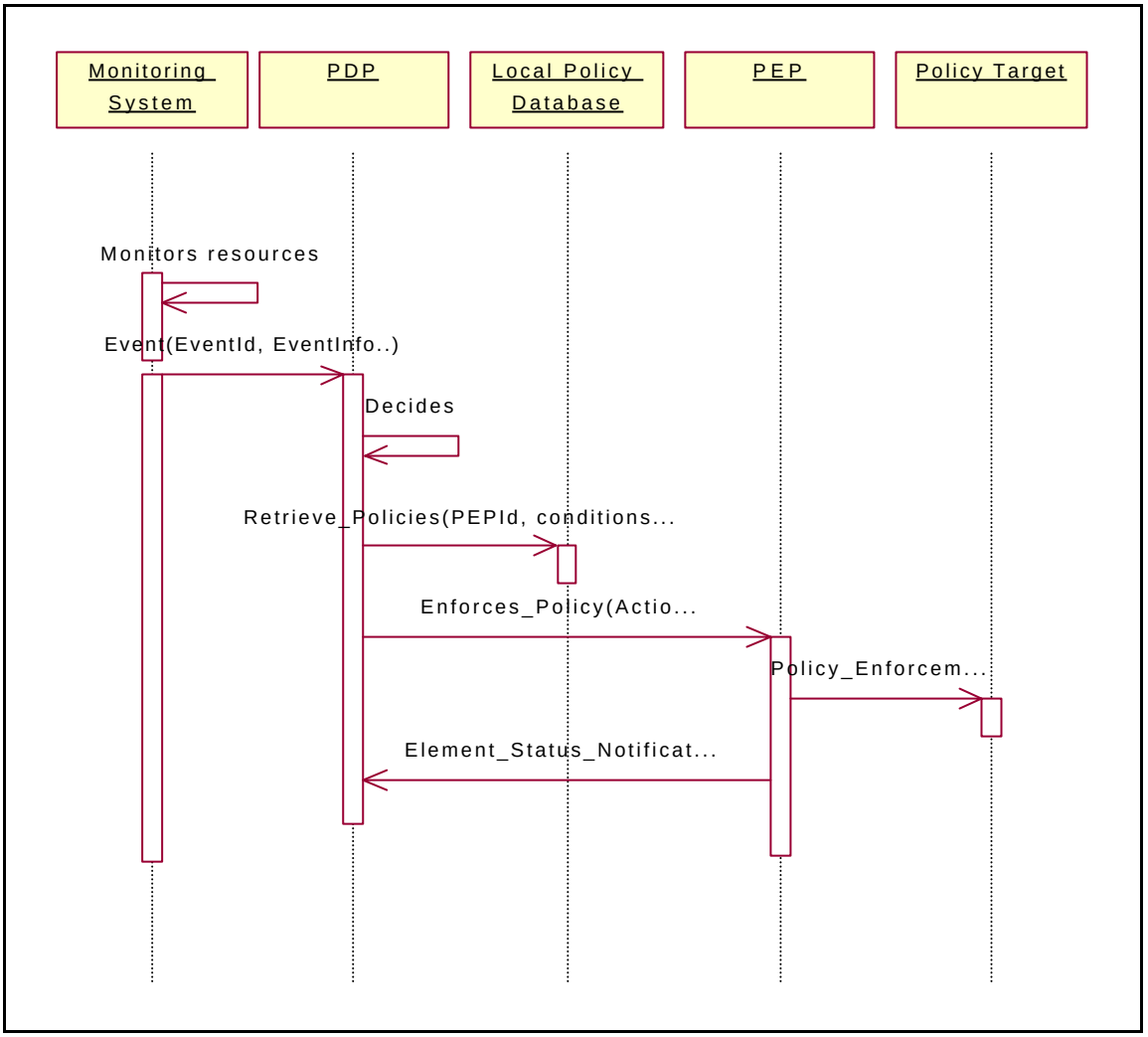

Figure 14 - Reconfiguration due to Network Status Sequence Diagram

2.4.1.3.2.3.2 Activity Diagram

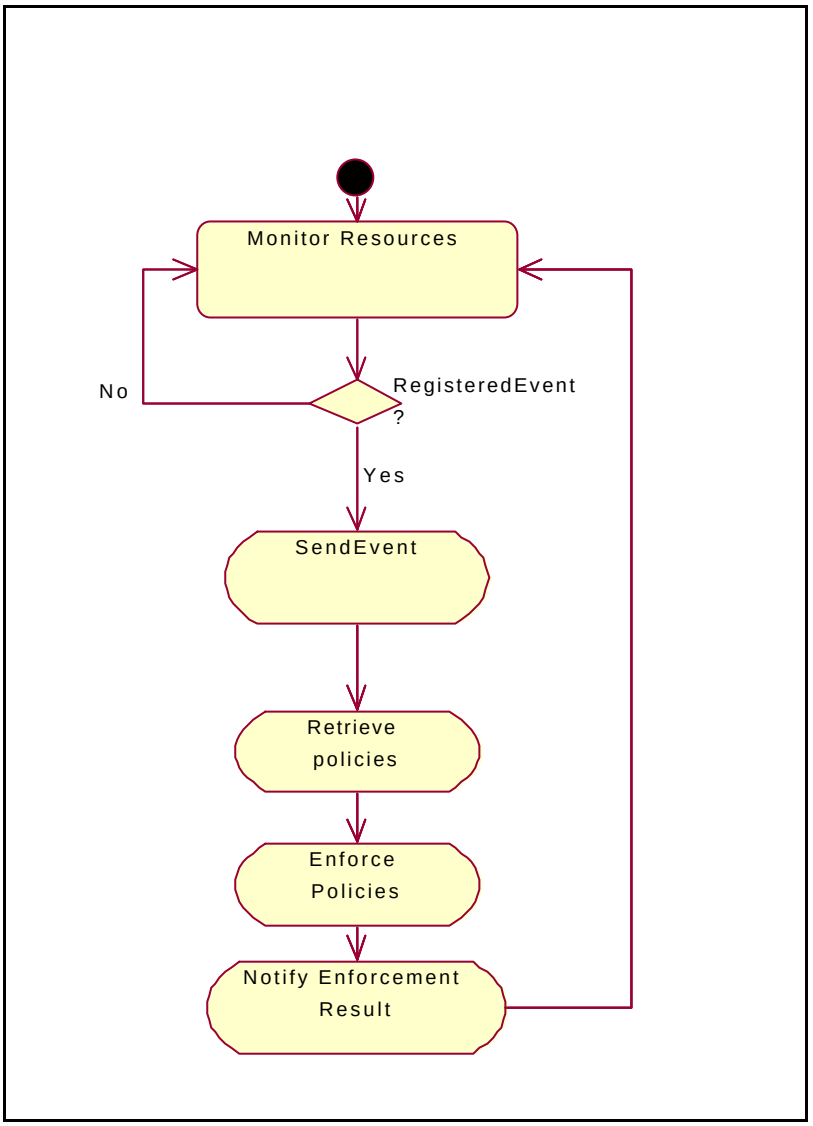

Figure 15 - Reconfiguration due to Network Status Activity Diagram

2.4.1.3.2.4 Application Provisioning Scenario

In this scenario we show how a customer can set their own specific policies whenever they have the authority to do itso.

1.- The application sends the policies to the policy-based management system along with a credential of the actor who wants to reserve the resources.

2.- The credential check sub-component checks if the actor with that credential is allowed to reserve node resources, and if so how many, using the policies database.

2.1.- If the credential is incorrect then an error message is sent back the application.

2.2.- If the credential is correct the policies along with the maximum allowed resources to be reserved are passed to the policy conflict check

3.- The policy conflict check component checks the consistency of the policies with the policies already set by this PDP, if any conflict is found it tries to solve it.

3.1.- If aconflict is found an error message is sent back to the application.

3.2.- In any other case, the policy conflict check passes the policies along with the maximum resources to the PDP.

4.- The PDP looks for the PEPs that will be in charge of the policies.

4.1.- If any of the PEPs is not found, then the PEP manager might ask to the ASP to download the PEPs.

4.1.1.- If the PEP could not be download then an error message is sent back to the application.

4.1.2.- In any other case, go to step 4.2

4.2.- The PDP should then install the PEP properly within the network element management architecture.

5.- The PDP then registers in the monitoring system the events to decide when those policies should be enforced, and stores the policies in the database.

6.- The monitoring service sends, when appropriate, the registered events to the PDP.

7.- The PDP will then ask the monitoring system the resources already reserved to that customer.

7.1.- If the resources already reserved plus the resources requested in the policies increases the maximum resources allowed an error message is sent back to the customer.

7.2.- In any other case the PDP retrieves the policies from the database and passes them to the appropriate PEPs.

8.- The PEP performs the correspondingent reservation actions.

9.- Finally, the PDP informs of any configuration change in the resources of the node due to the successful enforcement of the policy.

2.4.1.3.2.4.1 Sequence Diagram – Main branch: PEP previously installed

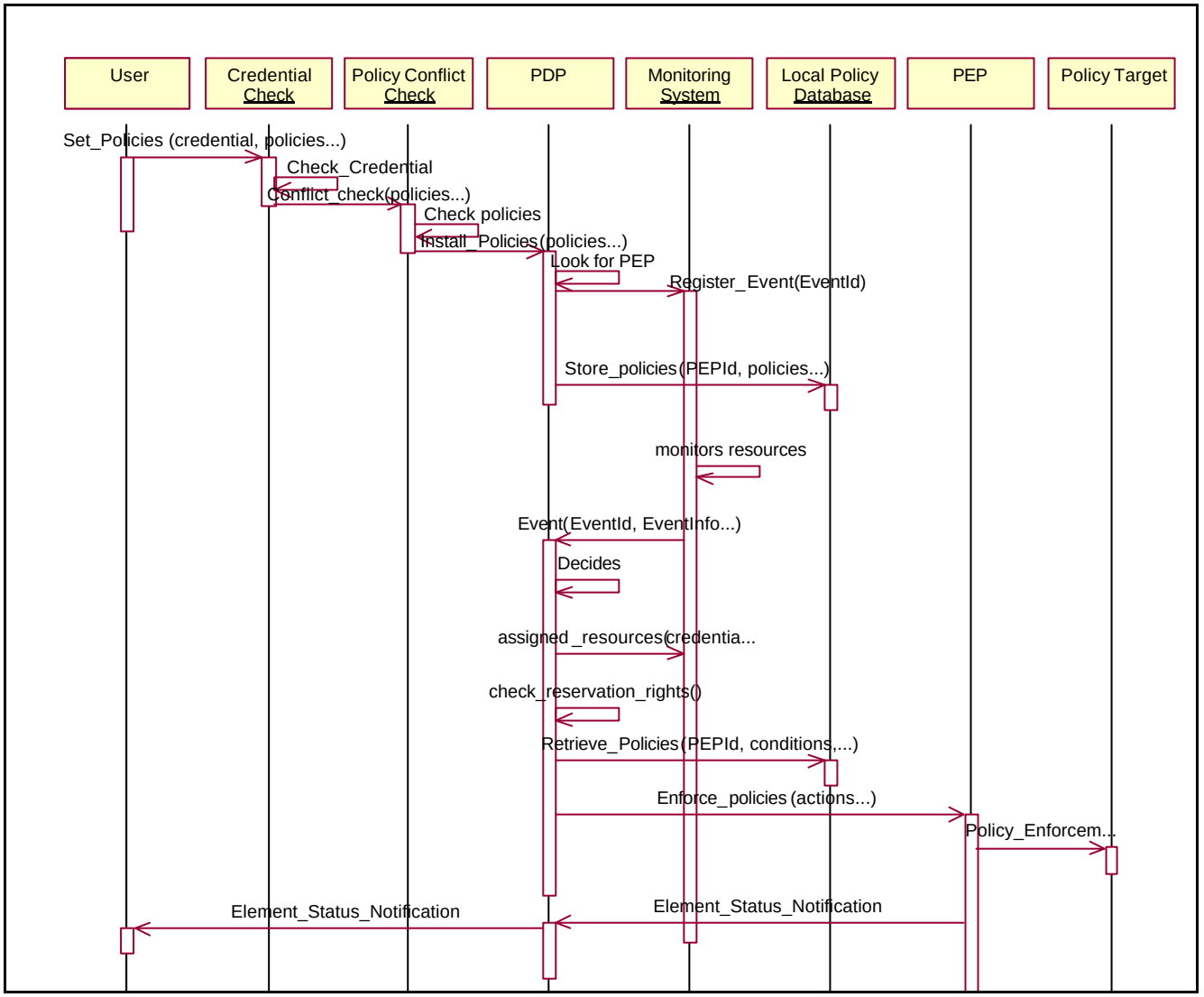

Figure 16 - Application Provisioning Scenario Sequence Diagram – Main branch

2.4.1.3.2.4.2 Sequence Diagram – Sub-branch: PEP Not Installed

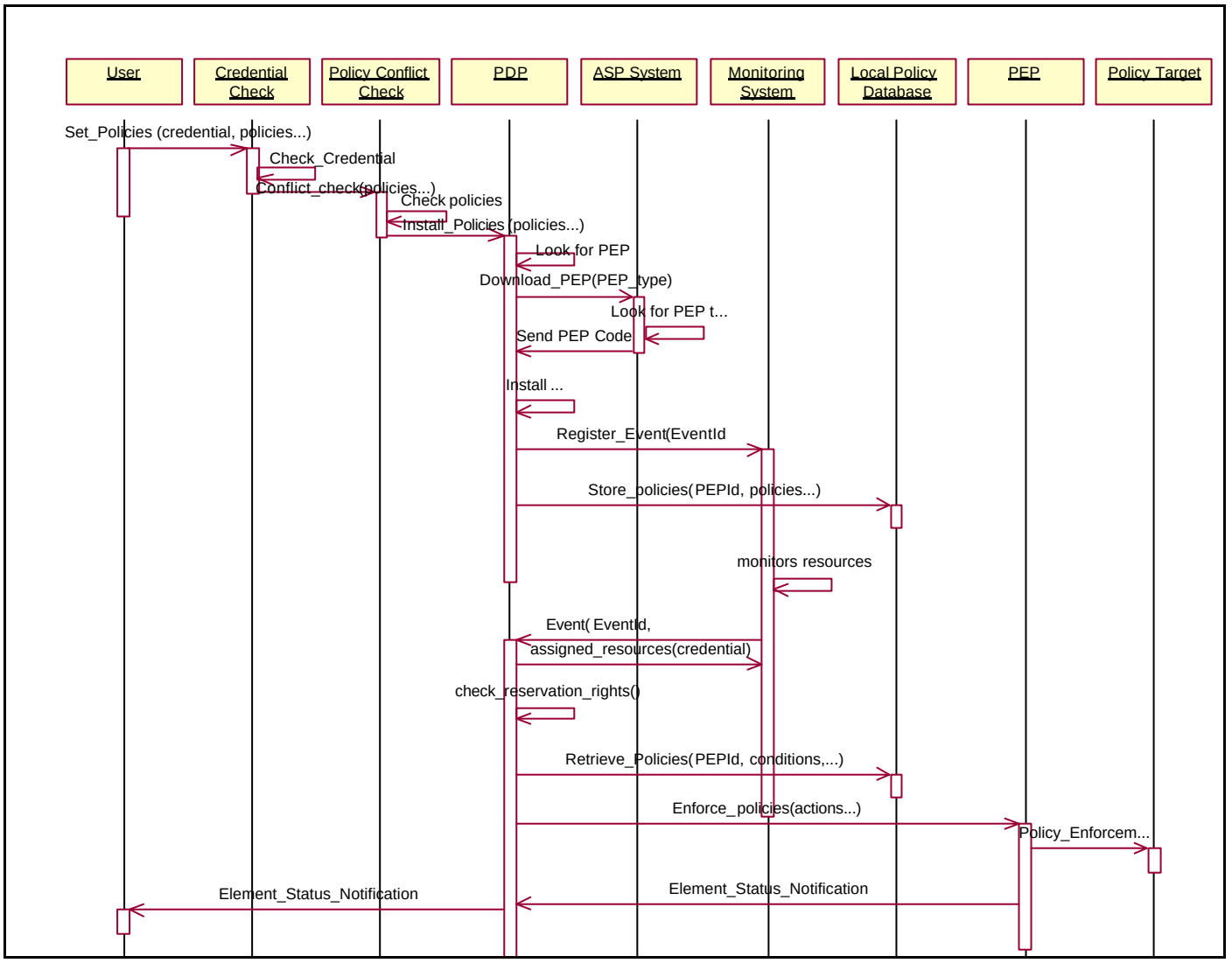

Figure 17 - Application Provisioning Scenario Sequence Diagram – Sub-branch

2.4.1.3.2.4.3 Activity Diagram

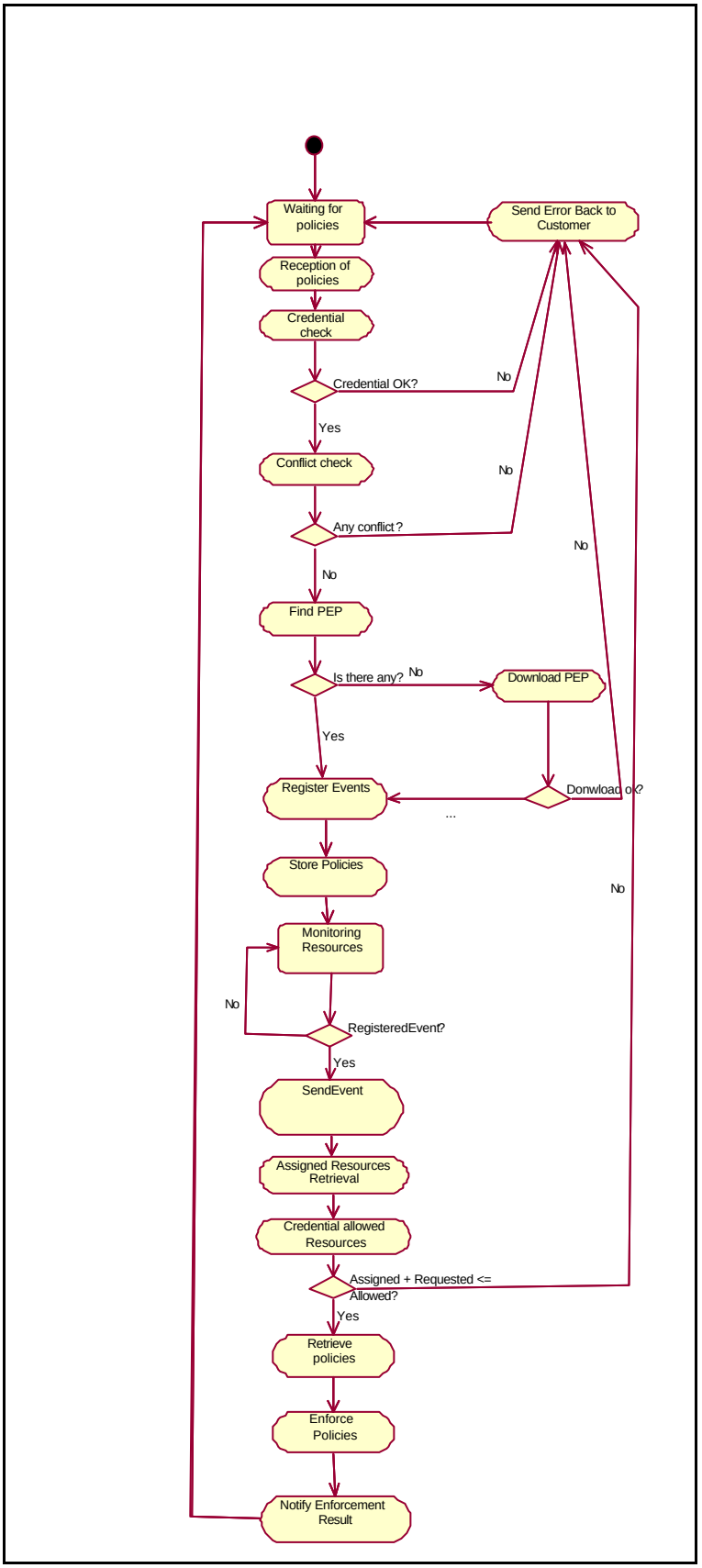

Figure 18 - Application Provisioning Scenario Activity Diagram

2.4.1.3.2.5 User-specific policies provisioning

The scenario shows how a user can use their own specific policies (and therefore their own specific PDP and PEPs) in order to manage their resources

Pre-condition: The appropriate meta-policies allowing the user to use their own specific policies should have been previously introduced in the Network Management System.

1.- The User introduces in the Policy-based Management system a policy, along with a credential, requesting the installation of their own management system to manage their resources.

2.- The credential check sub-component checks if the user with that credential is allowed to use their own management system, and if so which management functionality can they access, using the policies database.

2.1.- If the credential is incorrect then an error message is sent back to the application.

2.2.- If the credential is correct the policy along with the management functionality they are allowed to access are passed to the policy conflict check

3.- The policy conflict check component checks the consistency of the policy with the policies already set by this PDP, if any conflict is found it tries to solve it.

3.1.- If an unsolvable conflict is found an error message is sent back to the application.

3.2.- In any other case, the policy conflict check passes the policies along with the maximum resources to the PDP.

4.- The PDP receives the policy and the management functionality allocated to that user, and passes this information to the appropriate PEPs.

5.- The PEPs will then enforce the corresponding actions in order to ensure that the user-owned management system will only be able to access their allowed management functionality.

6.- The PDP will then request to the ASP the downloading of the User's management system.

7.- Once the code is downloaded, the PDP will inform the user of the successful enforcement of its policy.

2.4.1.3.2.5.1 Sequence Diagram

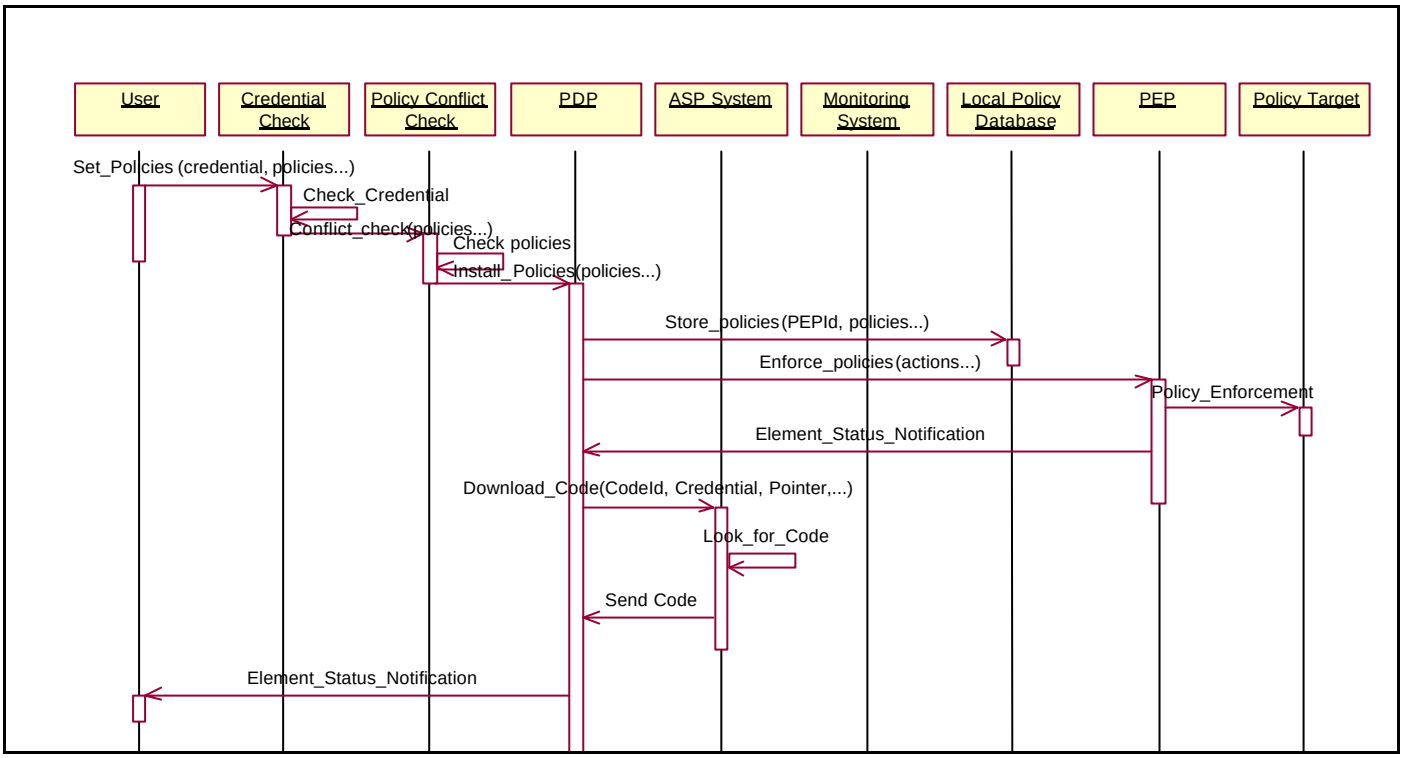

Figure 19 - User-Specific Policies Provisioning Scenario Sequence Diagram

2.4.1.3.2.5.2 Activity Diagram

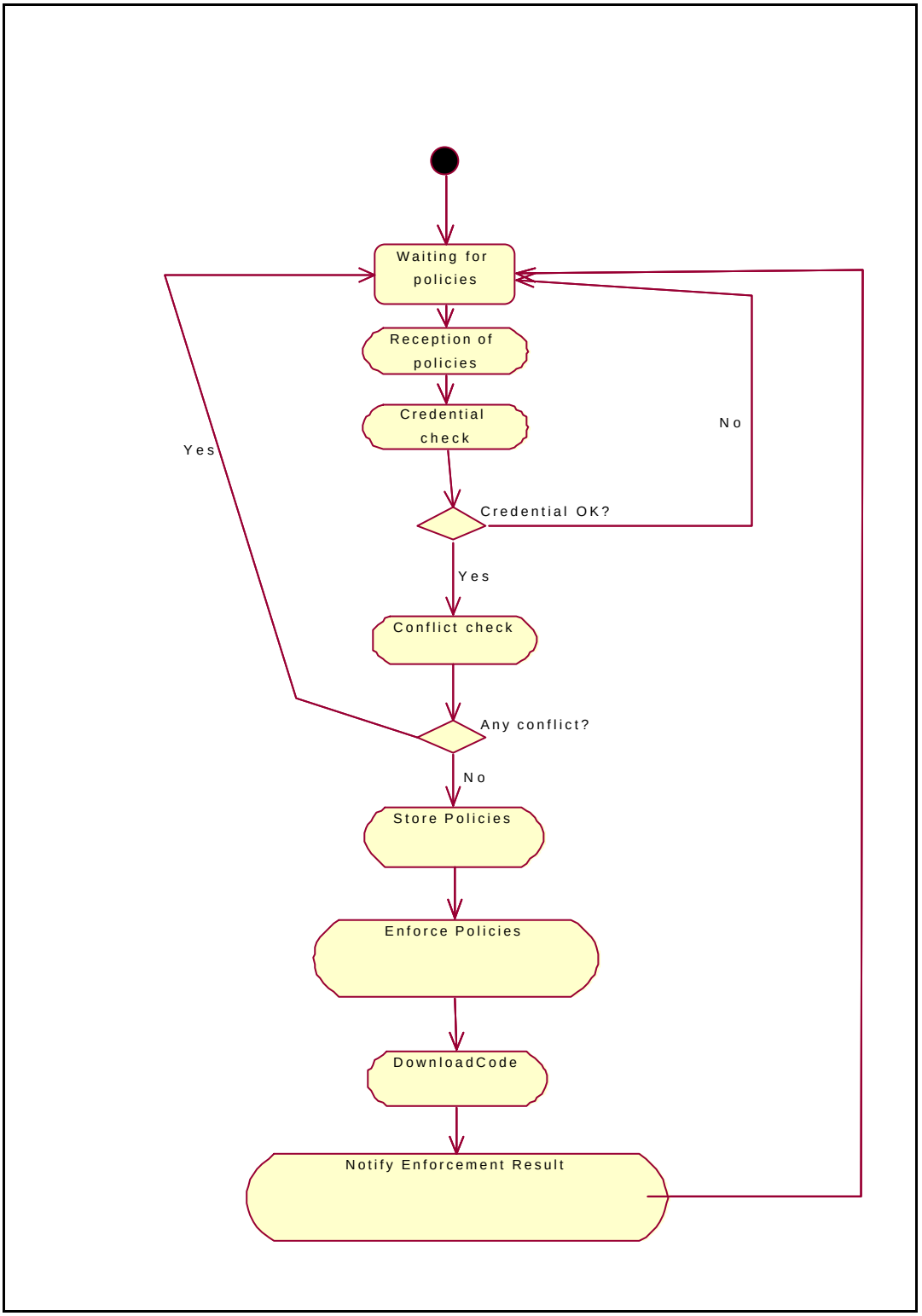

Figure 20 - User-Specific Policies Provisioning Scenario

2.4.1.3.2.6 Alarm Report Scenario

This scenario shows how the policy based active network management system described here can cope with one of the possible management requirements related to fault management: namelyalarm reports.

Pre-condition: The appropriate alarm report policies have been previously installed following the normal provisioning mechanism[3]. This implies, of course, that the appropriate PEP that treats these policies is also installed.

1.- An alarm situation is detected in one of the policy targets which results in the policy target sending the correspondingent alarm report to the monitoring system[4].

2.- The monitoring system then checks if any PDP has registered an event related with this alarm.

3.- If so, the monitoring system sends an event to the PDP.

4.- The PDP upon the reception of this event decides whether any policy should be applied.

5.- If so, it retrieves the policy from the database and passes it to the correspondent PEP.

6.- Finally, the PEP will construct and send the alarm report with the appropriate information.

[3] The syntax of these policies should be something like: if <conditions> then <alarm report>

[4] The policy target can also reflect this alarm by changing the value of an attribute. In this case, the monitoring service component will detect the alarm situation polling the value of the attribute.

2.4.1.3.2.6.1 Sequence Diagram

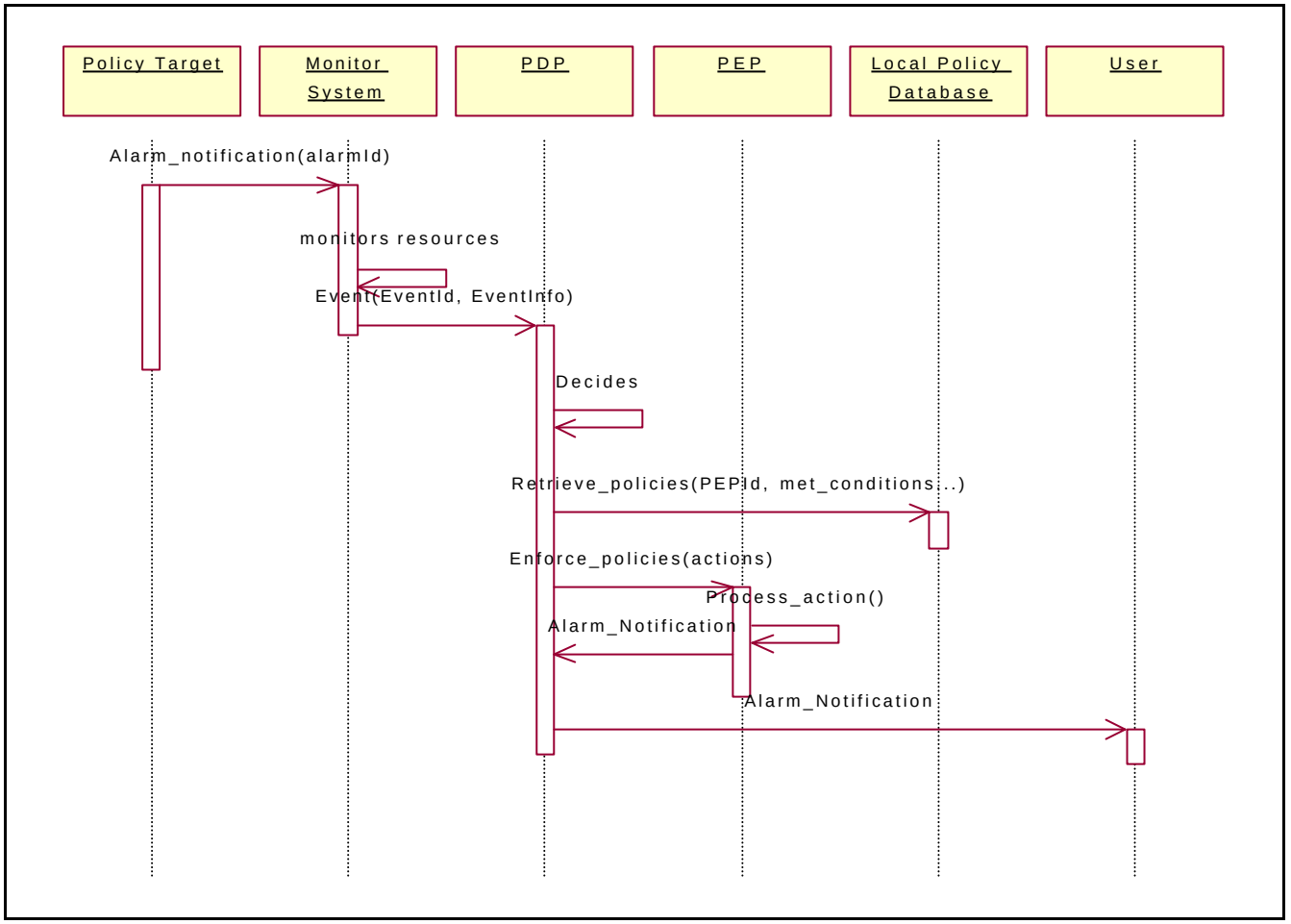

Figure 21 - Alarm Report Scenario Sequence Diagram

2.4.1.3.2.6.2 Activity Diagram

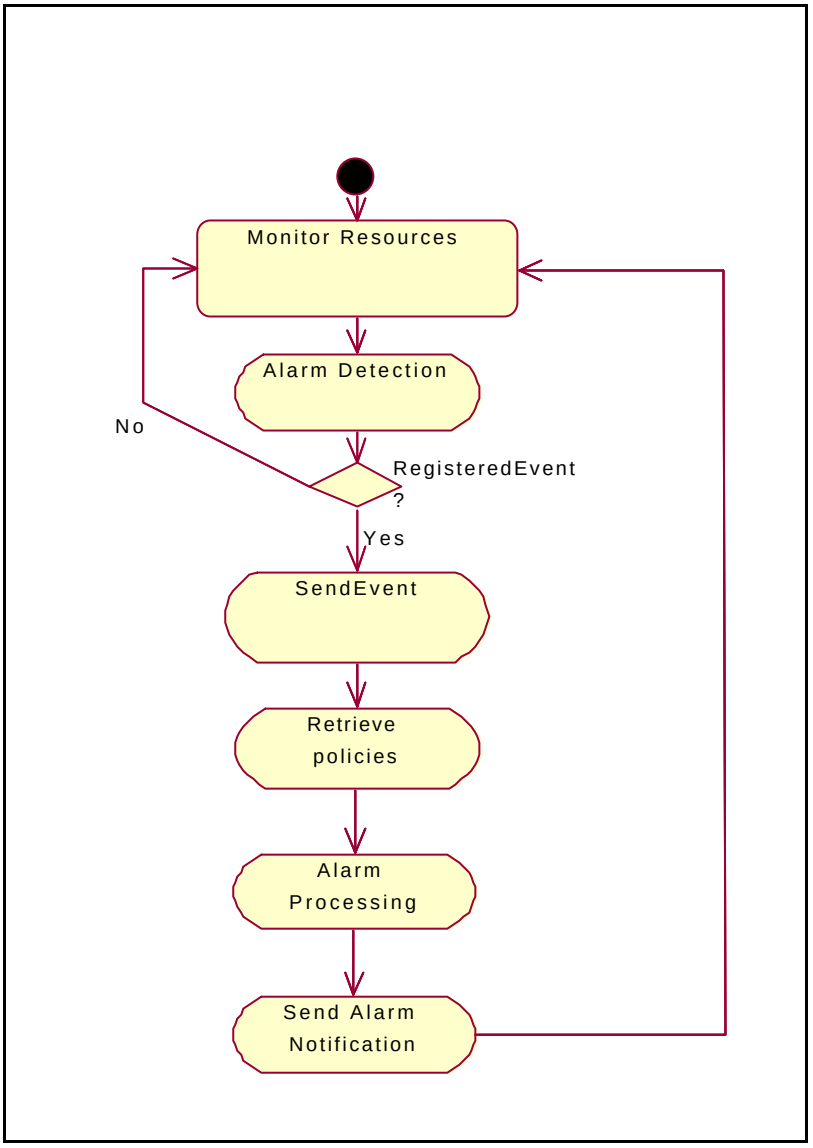

Figure 22 - Alarm Report Scenario Activity Diagram

2.4.1.4 Relation with the IETF Framework

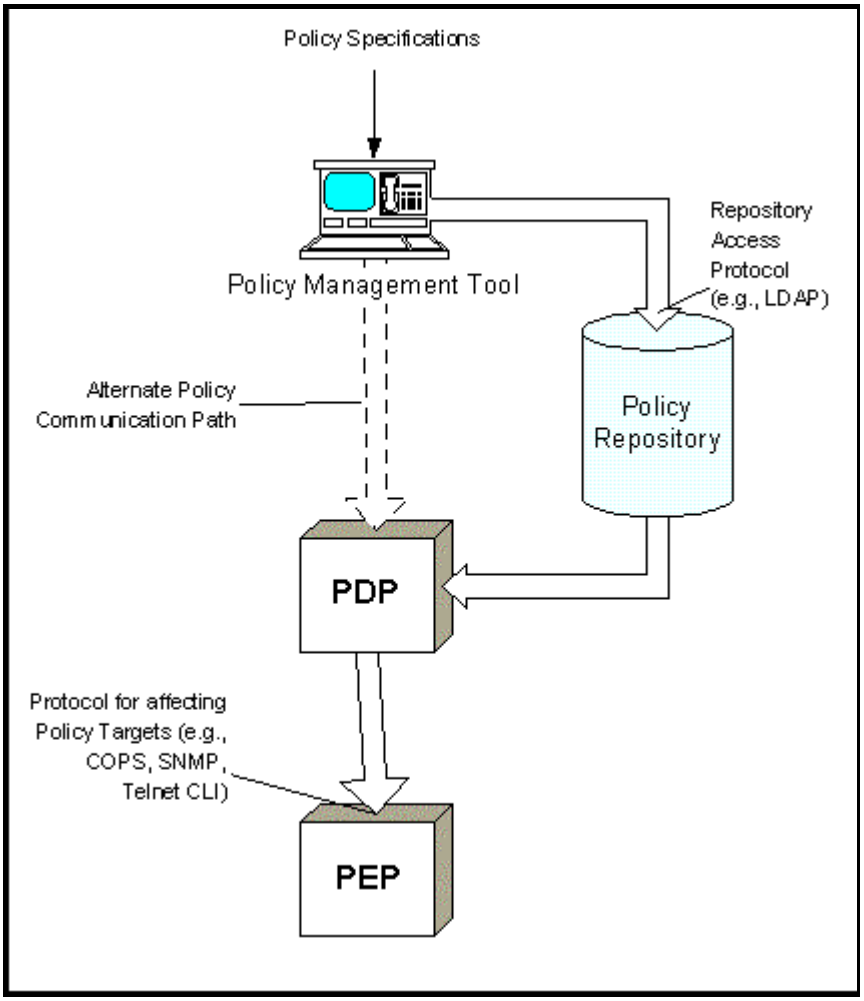

Figure 23 - IETF Policy Management Framework

With reference to Figure 23 above, the main components and modules in the framework are defined and the IETF proposal is briefly described:

Policy Rules: Serve as a point of interoperability between entities participating in any policy system within the framework. Policy rules are defined in a standardised information model by the IETF.

Policy Management Tool: Provides a user interface where the administrator can author or edit policies.

Policy Repository: Used to support reusability of data across managed objects; stored information aids consistency of information throughout the managed environment.

Policy Decision Point: A logical entity that makes policy decisions for itself or for other network elements that request such decisions.

Policy Enforcement Point: The point where the policy decisions are actually enforced. This is also a logical component that carries out action indicated by Policy Rules, hence implementing the functionality and behaviour specified in the Policy Rule. The Policy Enforcement Point is the device itself if the devide is policy-aware. Otherwise, the PEP is the entity in charge of receiving the policy decision and enforcing it in the device using commands understandable by that device.

Both the PDP and the PEP may be logically or physically the same. The choice is an implementation issue. There may be more than one PDP per PEP to enable continued operation in a failure mode, but only one at a time is to be the 'live' PDP.

At first glance (Figure 8), in comparison with the FAIN architecture, the most notable discrepancy is that the IETF does not explicitly define a security check component. The Policy Repository in IETF is analogous to the Policy Databases in FAIN, accessible at Int9. The PDP used in the FAIN architecture has basically the same functionality as that defined by the IETF. However, it has been extended to deal with some Active Networks specific issues, as application specific policies, reconfiguration due to network status, etc. Again, PEP functionality is basically that defined in the IETF framework always taking in count active networks specific issues. As foreseen in the IETF, the PEP interfaces with the managed resources at Int10, where the PBNM workgroup collaborates with the RCF workgroup. The PEP can be seen as the module that makes the managed devices policy-aware when they aren't.

More importantly, the IETF framework describes a more straightforward policy insertion process on the managed devices, whereas in FAIN, if a particular PEP is not found, the PDP first has to get the relevant PEP from the Active Service Provisioning database to be downloaded within the management architecture. With this in place, the policy from the local DB is downloaded and the PEP realises the policy-defined actions.

As to the decision of 'when' a policy needs to be realised, the FAIN PBM architecture also proposes a monitoring system to report registered events to the PDP in order to ease the decision making task. The IETF framework does not explicitly describe this functionality.

There are, as well, two main importants approaches of FAIN to Policy-based management that can not be seen when comparing with Figure 8, the two-tier policy based approach, and the dynamic extension of the management functionality capability of the architecture. Both issues have been described in detail when introducing the architecture. The two-tier policy-based architecture tries to solve scalability issues of the IETF approach that only considers one level (it was though to be used in LANs). The goal of the dynamic extension of the management functionality capability of the architecture is to face the Active Networks inherent requirement to the management system: adapting the management functionality to the managed resources that in Active Networks are inherently dynamic.

The IETF framework serves as a generic guideline for policy management and admission control. It does not specifically address in its design, management issues pertaining to active networks. As such discrepancies in terms of granularity of components within the architecture arises.

Policy Rules: Serve as a point of interoperability between entities participating in any policy system within the framework. Policy rules are defined in a standardised information model by the IETF.

Policy Management Tool: Provides a user interface where the administrator can author or edit policies.

Policy Repository: Used to support reusability of data across managed objects; stored information aids consistency of information throughout the managed environment.

Policy Consumer: This is a logical module that parses policy information, and translates Policy Rules into useable form by Policy Targets.

Policy Target: This is also a logical component that carries out action indicated by Policy Rules, hence implementing the functionality and behaviour specified in the Policy Rule. It is a specific aspect of a device or logical component. For example, E.g., a router has might have multiple interfaces, and each interface has multiple capabilities, hence if a particular router has 4 interfaces, and each interface has 4 manageable features, the router has 16 Policy Targets

Both the Policy Target and the Policy ConsumerPDP and the PEP may be logically or physically the same. The choice is an implementation issue. There may be more than one Policy Consumers per Policy Target to enable continued operation in a failure mode, but only one at a time is to be the 'live' Policy Consumer.

Policy TargetsPolicy TargetsFor each of the Policy Targets, a Policy Consumer is notified by thethe Policy Target

At first glance (Figure 8), in comparison with the FAIN architecture, the most notable discrepancy is that the IETF does not explicitly define a security check component (whether it be credential check or conflict check). The Policy Repository in IETF is analogous to the Policy Databases in FAIN, accessible at Int9. The Policy Consumer in IETF is closely in-line with the PDP described in FAIN. In fact, FAIN goes a step further by defining a PEP, which is in charge of each policy. The PEP interfaces with the Policy Target at Int10, where the PBNM workgroup collaborates with the RCF workgroup. The PEP can be seen as the module that makes Policy targets policy-aware when they aren't. The definition of the Policy Target by IETF is in-line with that of FAIN.

More importantly, the IETF framework describes a more straightforward policy insertion process on the Policy Target, whereas in FAIN, if a particular PEP is not found, the PDP first has to get the relevant PEP from the Active Service Provisioning database to be downloaded within the management architecture. With this in place, the policy from the local DB is downloaded and the PEP realises the policy-defined actions.

As to the decision of 'when' a policy needs to be realised, the FAIN PBM architecture also proposes a monitoring system to report registered events to the PDP in order to ease the decision making task. The IETF framework does not explicitly describe this functionality.

There are, as well, two main importants approaches of FAIN to Policy-based management that can not be seen when comparing with Figure 8, the two-tier policy based approach, and the dynamic extension of the management functionality capability of the architecture. Both issues have been described in detail when introducing the architecture. The two-tier policy-based architecture tries to solve scalability issues of the IETF approach that only considers one level (it was though to be used in LANs). The goal of the dynamic extension of the management functionality capability of the architecture is to face the Active Networks inherent requirement to the management system: adapting the management functionality to the managed resources that in Active Networks are inherently dynamic.

The IETF framework serves as a generic guideline for policy management and admission control. It does not specifically address in its design, management issues pertaining to active networks. As such discrepancies in terms of granularity of components within the architecture arises.

2.4.2 Specification of the Components of the System

t In this section three main areas of the Policy-based Management approach are discussed. These are, a detailed description of the FCAPS functionality supported by the architecture (see section 2.4.2.1). The focus in FAIN is in configuration and fault management, therefore these two sub-sections will be more detailed. The functional description is followed by an in depth description of the main architecture components in section 2.4.2.2. Finally, in section 2.4.2.3 the policy representation approach taken in FAIN is described. This section contains both the FAIN Information Model (partitioned in several domains of interest) which is based on the PCIM and PCIM extensions proposals([17] and [30] respectively) from the IETF; and the mapping of this proposed Information Model to XML and XML-Schema and [33].

2.4.2.1 Functional Capabilities of the System

2.4.2.1.1 Configuration Management

The configuration management functionality implemented in the PBM system will be basically, that offered by the PDPs and PEPs instantiated in the system. However,theresources offered by the policy targets limit the potential management functionality. Ideally, either the network operator or the users could manage all the resources.

The configuration management functionality could be carried out in three different ways: provisioning, signalling or self-adaptation. In the provisioning scenario either the network operator or the users send policies that should be enforced in the network to configure their resources. In the signalling scenario, a signalling protocol enabled policy target receives a reservation request (e.g. an RSVP packet) and passes this request to the element manager (this scenario is element level specific). Finally, in the self-adaptation scenario it is the same PBM system that reconfigures the resources after detecting a change in the network status.

The management system designed should be able to configure the resources of the managed network. These resources fall into two categories: communication and computational resources.

The communication resources are the bandwidth of links and the buffers/queues which store the packets while they are being processed and forwarded in the nodes. Using the PBM approach we want to assure that certain flows get the necessary quality of service, by allocating sufficient bandwidth and using priorities for the queues. This also means that the management system must be able to identify different flows based on certain fields of the packet header (source/destination IP addresses, protocol, source/destination ports).

The computational resources of the node are the processing power (CPU cycles), the memory and the disk storage of the Active Network Nodes deployed in the network. An active node may have multiple Execution Environments. These EEs are competing for the resources of the node and therefore the administrator must setup the corresponding policies to partition the computational resources of the node as needed. There is also a need to assign packet flows to EEs in concrete ANNs. The management system is also responsible for the creation, deletion and modification of EEs whithin ANNs deployed in the network. When the creation of a new EE is requested in a concrete ANN, the PBM system desgined should be designed to check existing policies to see if the user that made the request has sufficient privileges and if there are sufficient resources in that node for that EE. There should also be priorities for different EEs, to ensure that the most critical EEs always have enough resources to execute. These priorities also depend on Service Level Agreements made between the Network Infrastructure Provider and the Users.

In order to configure the devices, the Policy-based system must have an exact knowledge of the capabilities - and possibly the limitations - of the managed devices. This becomes more important in an active networking environment, where new services or execution environments may be dynamically installed in the network. New functionality and capabilities may be added to the active network, these changes have to be reflected in the management system.

2.4.2.1.1.1 self adaptationProvisioning Scenario

In the provisioning scenario, the User defines policies and sends them to the PBM System. The Security Checks component first examines if the policies have been defined by an authorized user and also checks the syntax and any possible conflicts with other policies.

The PDP gets the policies, after they have been validated by the Security Checks component. Before registering the policy conditions and downloading the necessary actions to the PEPs, the PDP should check if the policy targets of the policy have the necessary functionality to support its actions and conditions. This means that the PDP should maintain information about the capabilities of the policy targets. If a policy target does not have the capabilities to enforce a given policy, then an error message should be produced and the policy target should be enhanced. If the policy target can support the policy, then the PDP should also check if the existing PEPs have the required functionality. If not, a new PEP should be installed on the node.

The necessary information model is defined by the IETF, although it may require extensions to allow the configuration of AN-specific resources, like resource reservation for EEs, scheduling algorithms, assign priorities to certain EEs etc. The resources which goinghave to be managedable also depend on the FAIN Resource Control Framework (see Deliverable D2 [104], which should provide an interface to manage and configure AN resources.

2.4.2.1.1.2 Signalling Scenario

At the element level, resources can be configured using a signalling protocol. The signalling requests cannot be directly resolved by the PEP, because this component does not have the capability to make decisions on its own. So when such a request is encountered by a PEP, it is passed to the PDP, along with related information regarding the state of the PEP. The PDP then checks if there exist policies which can affect this request and sends the final decision back to the PEP.

2.4.2.1.1.3 Self-Adaptation Scenario

In the provisioning scenario, the administrator sets some initial configuration policies, or creates new policies after monitoring the network. The new policy is immediately downloaded to the PBM system and enforced, if needed. In the self adaptation scenario, the management system must be able to automatically set new policies, depending on the status of the managed resources, without requiring the manual intervention of the administrator.

The PBM system should be able to respond to certain events, or changes in the status of the nodes or the network. This can be accomplished by setting up the appropriate policies to define the response of the PBM system in such situations. Such policies however are distinct from other system policies, because when their conditions are true, they don't lead to the execution of commands by the policy targets, but to the deployment of new policies. This can be done by having an association between policies. However, this requires the modification of the policy core information model. Additionally, we can take advantage of the active network technology and request the dynamic injection of code, which will reconfigure the node to adapt to the new situation.

The format of these policies can be similar to that of the normal policies, however their conditions will be the status of the managed resources, or a particular event. The corresponding actions of these policies will include the creation, withdrawal or modification of another policy, or the installation of an active code module.

To evaluate the conditions of these policies, information (events) are required from the monitoring system. When the conditions are fulfilled, the PBM system must retrieve the corresponding policy from the database and execute its actions. This will result in the deployment of a new policy to the policy targets to deal with the changes in the status of the devices or the network.

2.4.2.1.1.4 Information model

A framework for the policy information model has been defined by the IETF. There also exist extensions to this protocol, mainly to support QoS services (DiffServ, IntServ). For usage in the FAIN Policy-based Management architecture, the information model should be extended to support active network specific capabilities such as computational resources and execution environments.

More extensive modifications to the Common Information Model will be needed to support the policies needed for the self-adaptation scenario. A new mechanism must be defined to allow the association between policies, so that one policy can control the deployment of another.

2.4.2.1.2 Fault Management

In a first classification, we distinguish two main fault types that should be considered when defining the management architecture:

- Critical resource faults: which limit the node or network operability and lead to to a network collapse and therefore have a severe impact on network behaviour. This kind of faults endangers the local fault management itself.
- Non-critical resource faults: which do not imply a serious damage in node or network working, so that the local management system is able to proceed with the control functions and autonomously recover from the failure. For example, this type of faults could lead to a performance degradation. Several policy types could be defined in such a case to give priority to certain activities, normally those ones that are essential to avoid network a collapse.

In both cases, it is necessary to implement the appropriate mechanisms for fault notification and local decision dissemination. In this way it is possible to modify the global policies that could be affected as a consequence of the local node anomalous behaviour.

Active networks also provide a means to enhance critical resource fault management since they provide the intelligence to foresee the predictable faults and take rapid decisions as required. It possibleeven to dynamically and autonomously modify the policies to reassign resources to promote a 'smooth' transition. Traditional network management imposes inherent delays that prevent this kind of procedures.

2.4.2.1.2.1 General fault management architecture

To tackle with the fault management issue in active networks we should consider four essential aspects. Firstly, the mechanisms that allow detecting the appearance of a fault in an active node or network should be established. Once detected, the second mission of the system is to notify such situation to the points in charge of its handling. Those points will include functions specialized in the fault correction which is the last task assigned to the fault management system.

The fault detection consists of two basic activities: the detection of events that warn about the rise of an abnormal condition and the correlation of these events to find out the cause, determining whether it might be considered as a fault. The distinction between the events that signal the symptoms of a problem and the root cause is suitable for the active networks case, since the active network itself has enough processing capability to identify the fault by noticing the symptoms. The event detection could be achieved following the polling model. For example, threshold events could be detected by checking the value of the relevant variables stored in the router MIB. Or the push model that would impose certain requirements on the policy target components. These components should be able to inject events asynchronously in the event detection layer.

The fault notification must be realized exclusively on the components in charge of resolving it, avoiding the neccesity of including filters in the managing components. A subscription model, similar to the one used in the CORBA event channel could be used. Those elements interested in the management of a fault will register themselves in the channel, so that they will be notified when its presence is detected.

In this way, several entities in charge of different fault relating tasks –such as starting corrective actions, storing the information associated with the fault, propagating the fault notification to other network nodes, etc- can coexist.

The local automatization of the fault management is one of the interesting aspects that arise from the joint use of a PBM fault management system with the active node capacities. The autonomous correction of a fault implies the specification of policies that point out the actions, which should be taken in each situation.

When the system is unable to re-establish a proper working of the node, it should disseminate the appearance of the error and the fault correction policies that have been applied.

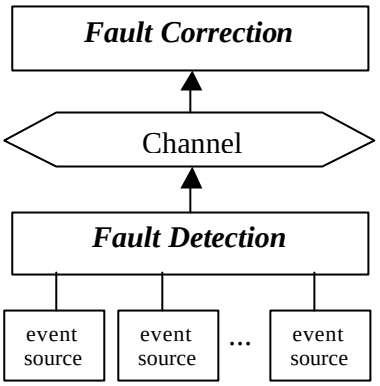

Figure 24 - Local fault management system blocks

Following, the distribution of the responsibilities between the different PBM architecture components will be analyised.

The event sources will correspond to what the IETF refers to as 'devices', understood in a wide sense as manageable hardware or software components. In the architecture we will follow the directive of isolating the fault management logic from the device specific characteristics.

At the element level,the managed router may offer us a MIB with valuable information, useful for the fault detection process. If the management elements incorporated in the NodeOS access to the MIB variables, they could become also event sources for the PBNM system. Likewise, it is necessary to know the APIs that the NodeOS offers for accessing to the self-diagnostic procedures available within the managed device.

The PEPs, responsible for the fault management policy enforcement, will take care of monitoring the specific events generated by its corresponding policy target, mapping them to events understandable by the system. Since the PEP has no global knowledge of the local events, they are not able to detect a fault, unless there exists a one-to-one relationship between an event and a fault.

The PDP will contain the fault treatment logic, being capable of taking corrective actions, modifying the active policies in different PEPs associated to the policy targets affected by the fault.

An event channel will be added to the local system. The PDPs interested in taking decisions regarding a concrete fault would subscribe to the channel to be notified of the fault rise. Several PDPs could be forced to modify the applied policies, which is the main reason to add the channel. The possibility of using a composite event channel as described in [50] will be evaluated.

In any case, the PDP will have to diagnose the fault. To help in this process an event database has been added to the system. This database will store the event history and could even be a means to obtain a basic event correlation adequate for demonstration purposes.

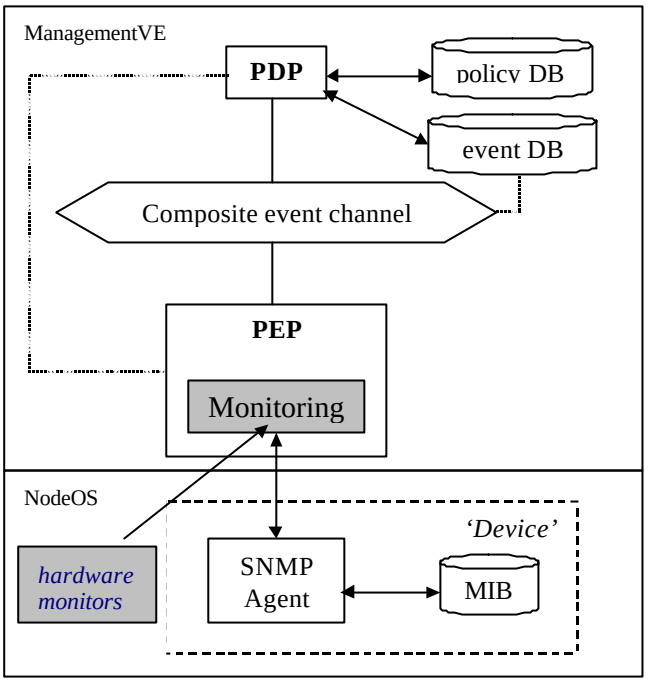

Figure 25 - Local fault management architecture

Another issue that has a deep influence on fault management is the definition of the fault management policies. The policies that seem to be more suitable are described in [51] as policy-policy target associations that contain new conditional associations with other policies. That is to say that the policy would inform of the conditions under which the new policies would be installed. As it is pointed out in [6], this would drive to a modification of the Common Information Model defined by the IETF, which will be undertaken during the implementation phase.

What seems to be clear is that the fault management policies should have a high priority, so that the corresponding actions will be preferably executed.

Detection as well as corrective policies will be defined.

2.4.2.1.2.2 Conclusion

The proposed architecture takes advantage of the PBM features (flexibility, extensibility) bringing them together with the active networks capabilities to provide an autonomous fault management system, which favours a quick solution to the problems, avoiding the failure to progress in its importance.

The specification of fault management policies is approached considering that conditions and actions both relate to policies currently active, e.g. a fault management policy could promote general policy updates based on the detected fault, the resources status and the active policies.

2.4.2.1.3 Performance Management

The PBM architecture described in this document bases its performance management functionality on the information obtained from the monitoring system and in the setting of adequate policies to manage the shared resource usage fairly. That is, a new customer would be able to reserve resources, as long as they have rights to do so and there are sufficient resources without affecting other customers.

Concretely, performance management functionality will be based on the monitoring policies domain, how these monitoring policies are to be treated by the architecture designed, and the monitoring facilities provided by the policy targets.

2.4.2.1.4 Security Management

The security management functionality developed in the PBM system can be divided into two main subtasks: controlling the access to management functionality itself and controlling the access to shared resources.

In the first case, it is the credential check sub-component that checks the credential given by the actor who wants to set a policy in the node in order to manage some resources. Whilst in the second case, it is the correspondingent PEPs that configure the resource access control layer to provide access to resources to customers. As such, security checks are done at both the management system, e.g. to check the credential supplied by the User along with the associated policies, and at the ANN, e.g. to check the credentials given by active code modules installed in EEs on behalf of the users, in order to access their assigned resources. Only the former is whithin the scope of the management system. However, security management is not whithin our focus, so the security and cryptographic algorithms necessary to check the credential will not be addressed by the management system.

2.4.2.2 Description of the System Components

2.4.2.2.1 Credential Check Component

The credential check component will be in charge of checking the privileges for specific active network management functionality granted to any actor.

Each actor that wants to fulfil the management functionality, such as VPN establishment, should also submit a credential. The Credential Check Component then takes this credential and looks in the meta-policy database for a meta-policy related with that credential. The Credential Check component retrieves the policy and checks if the intended management actions (policies) are available to the actor that presented the credential. Finally, if the credential is correct and the actor has the corresponding privileges these policies are passed to Policy Conflict Check component.

Since the working flow has been given previously, here we focus on the components and IDL-based interface.

The sub-components comprising the credential check components are depicteds as in Figure 26, whilst their detailed specification are shown in Figure 33.

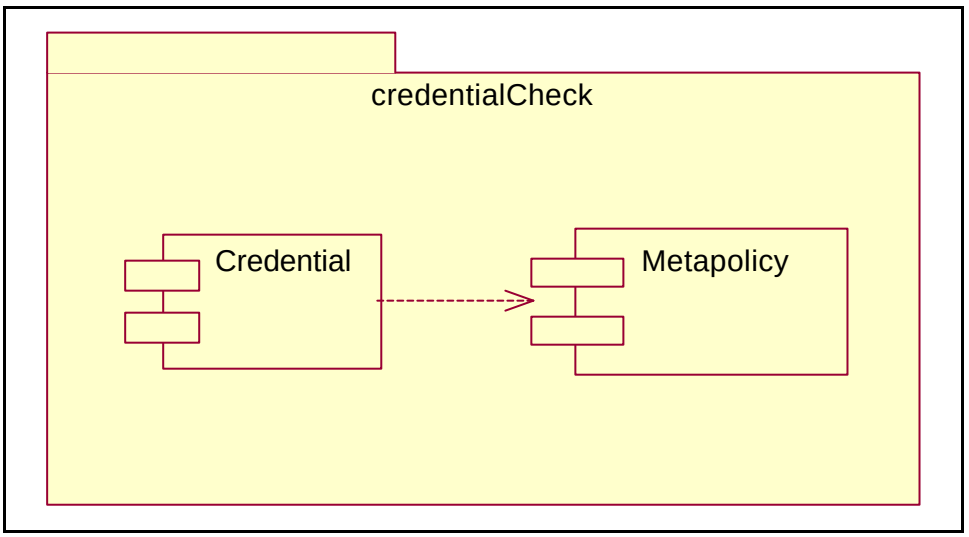

Figure 26 - Credential Components

The IDL presentations are shown in the section 2.4.3.1.1.

2.4.2.2.2 PDP Component

Within an IETF compliant PBNM architecture, the Policy Decision Point plays a fundamental role. document Here we present the main structure and functionality of that component defining and analysing its different subcomponents. The requirements that the active network concepts impose on the interaction with other components are highlighted.

The Policy Decision Point attends to two main aspects of the policy based network management. On one side, it takes charge of the retrieval of the policies coming from the User or from the active applications, proceeding to their distribution to the appropriate PEPs. On the other side, it determines the policies to be applied in every moment depending on its knowledge of the node status. A set of components has been identified as essential for the PDP to accomplish these responsibilities.

Once a policy has been transferred to the PDP, several actions are necessary in order to ensure that the policy is correctly deployed to the locations in which it has to be applied. Before actually proceeding to its distribution to the PEPs, the PDP should be sure that the policy could be correctly interpreted – i.e. it is syntactically correct- and that there are no conflicts with other policies currently running. Moreover, it has to be able to resolve such conflicts, which is an essential condition to provide support for application-specific policies.

Local conflict detection and resolution is one of the major functionalities of the PDP. Including the 'Conflict detection block' in the PDP is an essential condition since local conflict detection requires understanding the semantics of the policies[5] (conditions and actions). Since several PDPs could coexist in the same active node, distribution of policies among the PDPs is a prerequisite to detect the conflicts that affect each of them. Since not every policy has to be distributed to each PDP, different PDPs could have been provided with different sets of policies.

Since a PDP could be associated to one or several PEPs, to which certain kind of policies should be applied, a *name service* storing the PEP references must also be available. Further information regarding the PEP specific features could also be stored together with the PEP reference so that the PDP is able to provide the appropriate policies.

Locating policies in the policy database is another required function. This functionality is used both when analysing the policy conflicts and when examining the policies to be deployed to each PEP under demand.

The modules for parsing and interpreting the policies together with the decision logic blocks form the PDP core. Developing syntax independent policy parser interfaces could be helpful during the implementation process since the consolidation of the syntax selection may take into account several aspects not considered yet, that can only arise as the project evolves. In a more fine-grained vision, policy interpreters could be subdivided into *condition interpreters* and *action interpreters*. Both will use policy objects coming from the policy parser.

[5] Note that policies are dependent of the functionality being managed and so a policy could only be interpreted by its corresponding PDP.

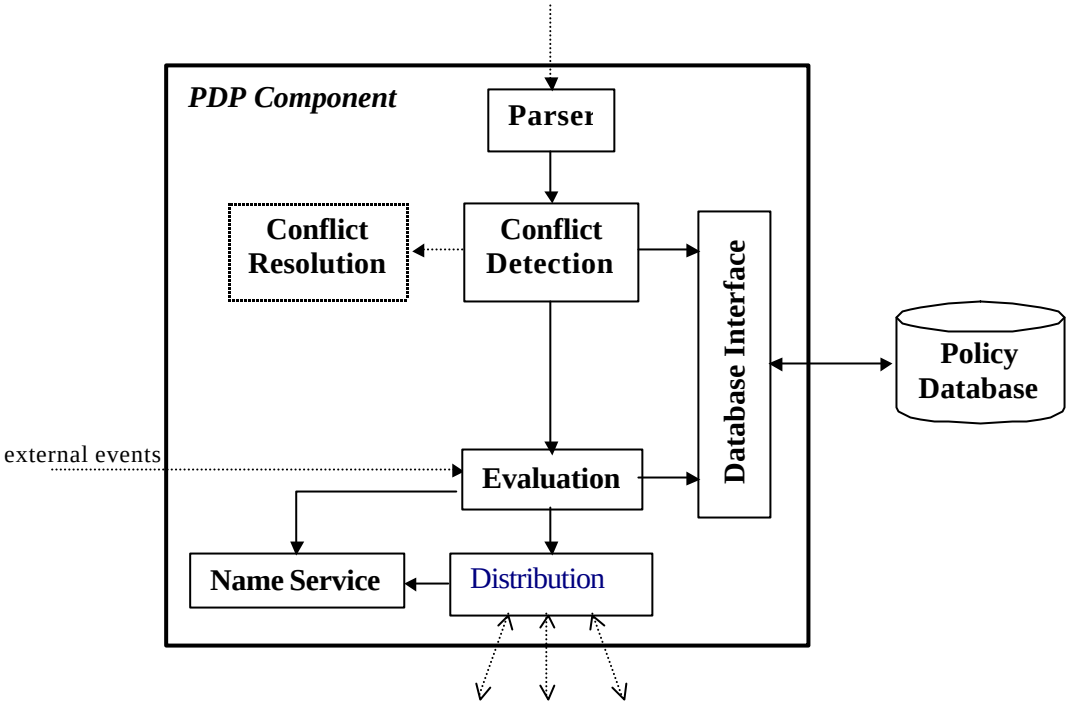

Figure 27 - PDP Internal Components

The functionality of each block is further described presenting the expected interactions with other inner modules.

2.4.2.2.2.1 Policy Parser

The policy parser is in charge of identifying and extracting the policy objects that constitute a policy. It also checks that there are no policy syntax errors, being a filter for policy acceptance. Since policy objects are going to be managed in several PDP subcomponents the parser should generate tree-like objects containing the parsed policy elements. Policy objects should provide navigation methods to allow accessing the information.

The parser must also provide methods for retrieving specific information contained in the policy without requiring creating the whole policy object. This feature would be specially useful during conflict analysis, when looking for concrete conditions or actions.

Provided that the policy information model could be extended, the parser should be designed so as to allow the extensibility of its parsing capabilities. Following a factory pattern would be a way to achieve such requirement.

2.4.2.2.2.2 Policy Conflict Check

Large distributed networks may well have policies that are specified by more than a single individual, possibly coming from various organisations. Authorisation policies specify what activities a manager is permitted or forbidden to do to a set of target objects while obligation policies specify what activities a manager must or must not do to a set of target objects [8]. Since a request may traverse through multiple domains governed by these various policies, conflicts of policies may arise

The policy conflict check sub-component is responsible for checking that the new policies that want are to be introduced in the management system do not cause any conflict with the existing ones. Two types of conflicts will be checked: syntax (or modality) conflicts and semantic conflicts. Syntax conflicts occur when the conditions are the same but the actions are opposite, i.e. "if <a> then <access>" versus "if <a> then <no access>". To realise this type of checking the policy conflict check component will need to access the local policy database in order to compare existing policies with new ones.

Semantic conflicts occur when two incompatible actions might be set at the same time. Metapolicies might help to perform this task by establishing which actions are incompatible and which are not, and to determine if the conditions for these actions might occur at the same time or not. If the metapolicy database and local policy database are physically different the policy conflict check component will also need to access the metapolicy database to be able to realise these tasks.

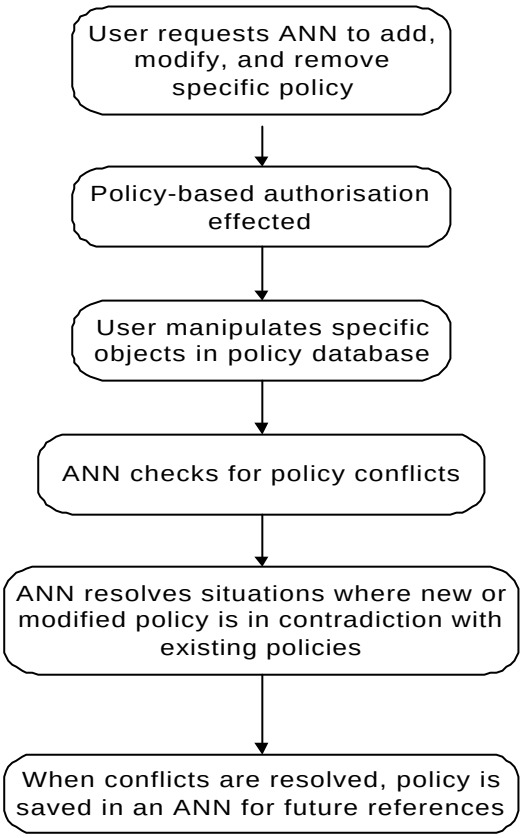

Figure 28 - Policy conflict check sub-component

2.4.2.2.2.2.1 Local Conflict Detection Block

This block is assumed to provide conflict detection functionality for application-specific policies. Particularly, at the element level, in fact, although policies coming from the Network Management Block have been previously checked for global conflicts they need also to be compared against the application-specific policies that only the active node is aware of.

The CDB (Conflict Detection Block) performs an evaluation of the conflicts that could arise between the incoming policy and other policies previously deployed promoting incompatible actions over the same devices. Once detected, two main strategies could be followed. The first one is the immediate cancellation of the deployment process, notifying the error to the entity that requested the operation. The second strategy is to resolve the conflict locally postponing the error notification until it is confirmed that it can not be treated by the PDP autonomously[6].

The block operation comprises several actions. First, the module must identify the PEPs to which the policy is being sent. After finding the actual policies currently running on those PEPs it is necessary to perform the conflict analysis. Finally, the result of the analysis should be reported.

Modality conflicts[7] would be recognised by inspecting the policy type but detecting semantic conflicts requires the knowledge of the policy fields meaning and possible values [9]. Therefore, conflict analysis will need additional information[8] about the policies themselves.

The conflict check interface should check for static conflicts derived from Policy Rules whose conditions are simultaneously satisfied, but whose actions conflict with those of currently existing rules [6]. An important point to note is that rules may be 'time-based' (specifying an effective validity period in the future) or based on dynamic state information. These rules may indeed conflict with others. However, these conflicts may only be detected at the time that the rule becomes valid and enforcement actions are attempted.

2.4.2.2.2.2.2 Local Conflict Resolution Block

This module is in charge of resolving the local conflicts that appear when trying to deploy application-specific policies. Applying precedence on the policies is one means to resolve certain kinds of conflicts. Another way is to make use of the metapolicies that would specify the set of actions to be taken to resolve each kind of conflict. The last one is the more extensible mechanism but requires a complete metapolicy framework to be developed.

2.4.2.2.2.3 Database Access Interface

The Database Access Interface main aim is to provide a simple query interface. Efficient policy database searching functions are critical to avoid a penalty due to unnecessary database accesses. Several components will ask for policies that possess specific conditions, being the responsibility of the module implementing the interface to locate such policies in the database, retrieving them and returning them in an appropriate format. Therefore, the module needs to know the structure being used to store the information.

---

[6] The first approach is simpler to implement while the second is more powerful. It should be taken into account that in the first stages it is preferable to implement the basic functionality and so we prefer the first strategy to develop an application prototype.

[7] Examples of modality conflicts include inconsistences between policies expressing authorisation –or unauthorisation- to perform an action and policies expressing obligation of doing –or not doing- that action. For example, a policy might oblige a PEP to perform an action it is not authorised to, because of the constraints imposed by another policy.

[8] which could be provided in the form of metapolicies.

2.4.2.2.2.4 Policy Evaluation Block

The Policy Evaluation Block takes the responsibility of deploying the policies as necessary. It consists basically of a Decision Making module and a Policy .Interpreter, which are analysed further in the following paragraphs.

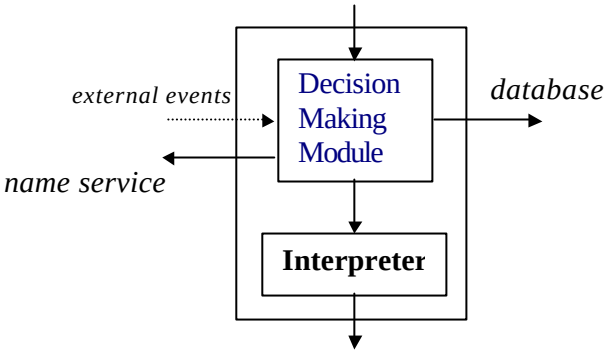

Figure 29 - Policy Evaluation Block components

2.4.2.2.2.4.1 Decision Making Module

In this document, a *decision* is considered a means to adapt the set of policies installed in a PEP to the circumstances. The PEP will only require a decision when its configuration changes in a way not covered by the installed policies [11].

In such a case, the decision-making module is responsible for evaluating the node status and decide whether there are high-level policies that fit the situation, and should be taken to the Policy Interpreter. For this purpose the policy data received from the PEP (for example, its current policy configuration) will be taken into account. The PDP should also have enough knowledge of the node status and capabilities to avoid delivering policies that cannnot be implemented, for example, due to resource unavailability.

This behaviour can be triggered not only by PEP requests, but also because of external events received by the PDP by any other means (for example, collected from the event channel). In this case, a call-back interface is used to be notified and acquire the information associated to the event.

2.4.2.2.2.4.2 Policy Interpreter

The policy interpreter becomes a core evaluation component, being in charge of translating the high-level policies into appropriate policy objects for its actual enforcement. In a finer-grained view, the policy interpreter will consist of a condition interpreter and an action interpreter. Among the condition interpreter functionalities, its ability to evaluate whether the rule conditions may be treated by the PEP itself or not, is a main requirement. In the former case, translating them according to the condition evaluation mechanisms would be realised either by creating policy objects as defined in the corresponding Policy Information Base or by including specific parameter values into previously existing policy objects. In the later situation, the Decision Making Module would take the responsibility of evaluating the policy conditions based on the corresponding low-level conditions returned by the interpreter.

The action interpreter will execute the specified actions would receive the high-level policy actions and would map them into the corresponding PIB Policy Classes, creating the appropriate policy objects. Thus, to correctly interpret a policy, appropriate condition and action interpreters should exist for each kind of condition and action included in the policy.

In order to tacklewith policy criteria noteven known at PDP design time (e.g. those coming from active applications) we intend to use an active approach. The active packets containing the -level policies (data) will also contain the condition or action interpreters (code) required to execute the policy which are not in the system yet. These interpreters will be treated as the pieces or "building blocks" of the policy interpreter. This way, the PDP can acquire knowledge about the policy classes used by the PEP.

Using this scheme, the same PDP type can be used for managing different types of enforcement clients.

2.4.2.2.2.5 Name Service

The PDP needs to keep track of the PEPs that it is serving. To issue this requirement the name service will provide information about the PEPs location and type. This service will be used to choose the policies that are to be delivered to each PEP depending on its role. The name service interface should contain methods to insert PEP information, and to retrieve a list of PEPs matching a certain role or type. Whether to use internal or external[9] name services is to be considered.

The CORBA name service interface definition could be used to support the interactions described formerly. The PEP role and associated target would be used as *Naming Contexts* to structure the information. At the leaves of the tree we would found the PEP identifier. Therefore, the PEP references would be obtained resolving the *Name* that contains the required role.

The *Name Components* included in the *Name* would contain the PEP identifier in the *id* field whereas the *kind* attribute could contain additional information about the PEP functionality.

The *list()* method can be effectively used to retrieve the list of PEPs matching a certain role, and the *BindingIterator* interface provides the means to browse the list. Refer to the *Interoperable Name Service Specification* [57] for more information on these methods.

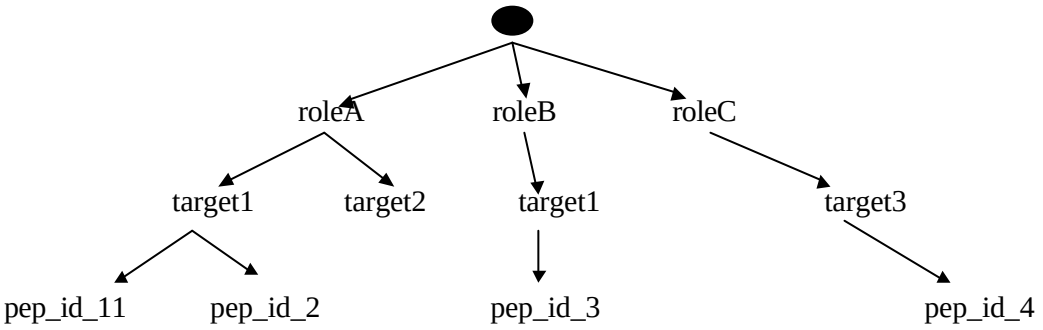

Figure 30 - Hierarchical naming structure

2.4.2.2.2.6 Policy Distribution Module

The Policy Distribution Module will be responsible for the PEP admission and dynamic binding functionalities. Once bound, it interfaces with the PEPs, distributing the policy objects in the agreed format. Therefore, this module could be considered as implementing the COPS-related logic functionalities.

The policy objects that come from the Policy Interpreter do not have a specific destination, but the role or type of enforcement clients to which they apply. It is the policy distribution module mission to find the PEPs playing the selected role and proceed to actually transfer the policy components. For this purpose, it will access the name service.

[9] Such as the CORBA name service.

2.4.2.2.2.7 Conclusions on PDP Architecture

The PDP architecture presented in this section conforms to the IETF specifications, but it also extends the basic PDP capabilities by the use of active code attached to the policies being distributed. This way PDPs provide the framework to make decisions based not only on the policy data but also on external knowledge carried in active packets.

Note that it is assumed that PEPs are able to interpret the policies delivered to them. Also, it is expected the PEP will store the set of policies to manage a given configuration. As a consequence, the proposed model does not compromise the active node performance since the decision processes do not run frequently.

2.4.2.2.3 Monitoring System

In order to define a highly integrated monitoring service, we propose the use of specialized policy-controlled monitors. These monitoring enforcement points interact with the policy target metering components and other facilities to extract the useful information which is defined in the configuration provided by a set of policies.

The objective of this architecture is to take advantage of the knowledge PEPs have about the devices they control. So monitoring information is collected, not only from specific monitoring PEPs but also from every PEP included in the system.

Event propagation is one of the key advantages of this architecture. Using an event channel allows information collected by a PEP to be distributed to PDPs interested in the events, although they could even not know of the existence of such a PEP.

The proposed monitoring service adapts to a policy-based architecture. It has been realised that the capabilities offered by the PBNM systems allow to takeing advantage of the existent framework for monitoring purposes.

The components directly involved in the node monitoring functionalities are depicted in Figure 31. The policy target will provide a set of metering blocks, as well as access to the statistic functionalities and MIBs included in the router (for the element level). The mechanisms to interact with the hardware monitors will be device specific and therefore should be isolated from the higher layers of the PBNM system. Following this directive, monitoring enforcement points allow to configure and collect the information provided by the metering blocks. Several parameters are to be adjusted, such as the polling interval or the event reporting frequency.

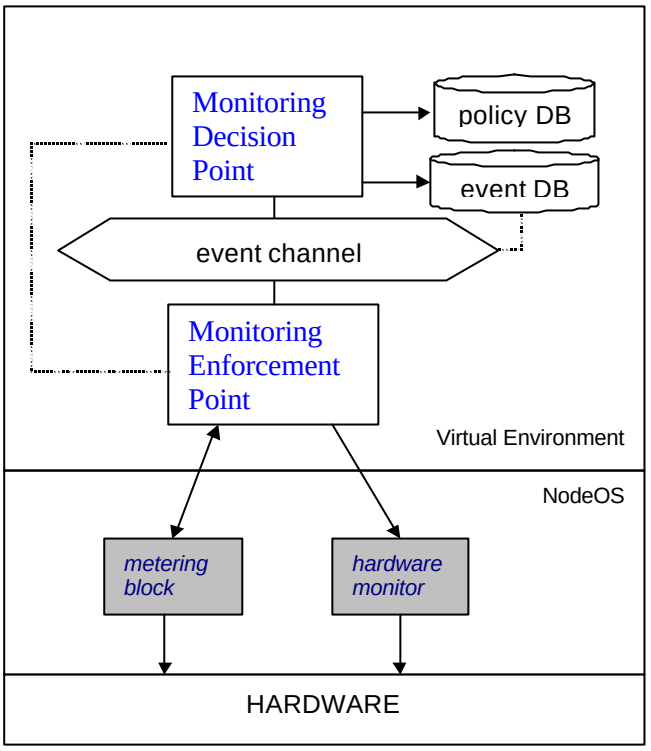

Figure 31 - Monitoring system architecture

All actions over the metering blocks are to be expresed on via monitoring policies (e.g. increasing or decreasing the polling interval depending on the device status). Specialized monitoring PEPs will be in charge of enforcing such policies on the policy targets they are assigned to. Also, general PEPs can act as a monitoring information source as part of their normal operation.

PEPs deliver the monitoring information to the PDPs through the defined PDP-PEP interface, which supports its transference either synchronously or asynchronously. General PDPs will use this information to create a picture of the resources status in order to make accurate decisions.

Monitoring decision points will be responsible for deploying the monitoring and notification policies to the specialized monitoring enforcement points. They must know the list of available monitoring elements and the type of resources under control.

Event capture and propagation is another of the PEP functions. Once an event coming from the metering blocks is detected, the PEP maps it to an appropriate format and delivers it to an event channel. PDPs will subscribe to the event channel waiting for specific event types and will provide a callback interface to the channel in order to receive notifications. This behaviour is appropriate since several PDPs could be interested or be affected by such events, even although they came from PEPs that do not relate with them. Those events signal resource status changes, which in turn are the key to determine whether the conditions for policy deployment are met.

The PBNM concepts and framework can be reused to develop a monitoring architecture. Using this approach, device-specific features are isolated from the rest of the system. At the same time, the PBNM system is enhanced with the addition of an event channel that helps the PDPs to be notified of the events they are interested in.

The monitoring system takes the advantages associated with the active network PBNM, including the extensibility through the addition of new monitoring components as required (for example, to monitor an application).

2.4.2.2.4 PEP Components

The Policy Enforcement Point component has the responsibility for the correct enforcement of the system policies. However, the PEP is not involved in the decision process. The decisions are made by the PDP and then they are downloaded to the PEP. The PEP also has to communicate with the policy targets, in order to enforce the actions that derive from the policies.

The PEP will have the following components:

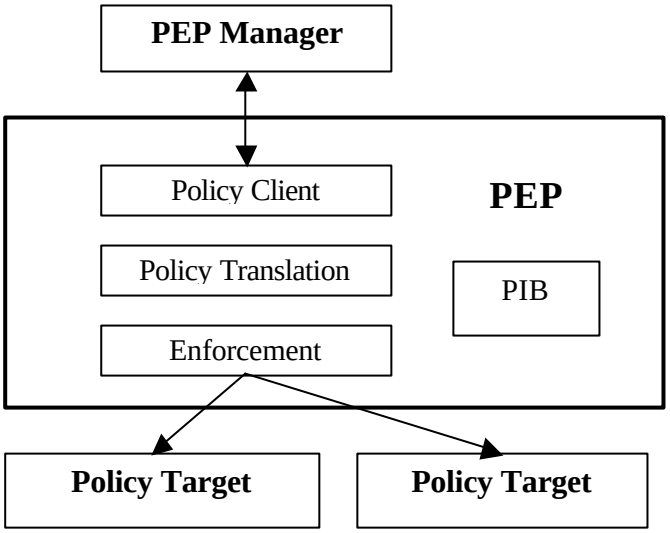

Figure 32 - PEP Internal Components

2.4.2.2.4.1 Policy Client component

This component has the necessary functionality, to support the communication with the PDP. It implements the client interface for the PEP-PDP communication.

2.4.2.2.4.2 Policy Translation

The policy decision received from the PDP will be in a format that may be not understandable by the Policy Targets, for example when the PEP functions as a proxy for a policy unaware device. This component will convert the decisions to a set of commands, according to the API provided by the underlying policy targets.

2.4.2.2.4.3 Enforcement component

This component communicates with the policy target. It actually enforces the policies, by using the appropriate target operations. It also has a discovery mechanism, by gathering the capabilities of the underlying targets, which will be reported to the PDP.

2.4.2.2.4.4 Policy Information Base (PIB)

This base stores the capabilities of the PEP component, which depend both on the underlying policy targets as well as the functionality coded into the PEP. It can also store the policy classes downloaded by the PDP to the PEP. The PIB can also be accessible by the PDP. The Policy Information Base can be structured according to [13]. A framework PIB is also provided in [14]. This model will probably require extension, in order to be used in the FAIN framework.

2.4.2.2.4.5 PEP functionality

The functionality supported by the PEP can be the configuration of a policy target, the enforcement of security/access policies or the reservation of resources. To accomplish this, the PEP should know the interface of the policy targets, as well as the capabilities of the underlying managed elements. This information should also be sent to the corresponding PDP, so that it knows the conditions and the actions which can be supported by a particular policy target. In this way the PDP will be able to check if a certain policy can be applied to the policy target.

2.4.2.2.4.6 Reporting Device Capabilities

The capabilities and the limitations of a policy target determine the policy functionality that can be supported by this policy target. A PEP which is responsible for a policy target should report these capabilities to the PDP, so that it can check if a new policy can be deployed to a specific policy target. So when a session between a PEP and a PDP is initiated, the PEP also passes the information related to the capabilities and limitations of the managed element.

These capabilities and limitations depend either on hardware specific issues of a managed element or on statically configured parameters. However in active networks, where services can be dynamically installed, these capabilities may also change. For example, an active node may initially not support DiffServ, but a basic DiffServ service can be installed later. In this case the PEP should also report this change in the capabilities of the node, something that could not happen with a traditional node.

2.4.2.2.4.7 Downloading Policy Decisions

The PEP does not have the ability to evaluate policy conditions. This task has to be done exclusively by the PDP, which uses known state information of the PEP as well as data obtained from the monitoring service to make the decision. Then the policy decision has to be downloaded to the PEP.

The enforcement of policies can be accomplished in two ways:

- Initiated by the PDP. The PDP, using the monitoring service, finds that the conditions of a policy are true. It retrieves the corresponding policy actions from the policy database and downloads the relative decision to the PEP.
- Initiated by the PEP. Usually this happens in the case of an incoming signalling request. The PEP receives a reservation request, but it is not capable of making a decision on its own. Therefore the request is passed to the PDP, along with information about the state of the PEP, which will help the PDP to reach the decision. The PDP evaluates the request and downloads the decision to the PEP. Optionally the PEP may include a Local Policy Decision Point (LPDP), to evaluate certain policies locally. However the final decision must be made by the PDP and it may override the one made by the LPDP.

It is possible that an existing PEP may not be able to enforce the policies on a target. This is likely to happen in an active network, since new services can be deployed dynamically and new types of EEs can be installed in the nodes. In this case a PEP with the desired functionality (or even a PDP-PEP pair) should be dynamically installed. This can be done using the Active Service Provisioning infrastructure, part of whose functionality is to act as a code distribution mechanism.

2.4.2.2.4.8 Policy Enforcement

After receiving a policy decision from the PDP, the PEP proceeds with the actual enforcement of the policy. The policy targets will execute the actual commands that implement the policy actions. The PEP should be able to convert the decision made by the PDP to the necessary commands for the policy target. This is dependent on the interface offered by the policy target. For example if the policy target is a traditional router, the PEP might use the necessary SNMP operations to configure it.

2.4.2.2.4.9 Policy Enforcement Feedback

After the enforcement of the policy decision the PEP should provide feedback to the PDP about the result of the actions taken, if the policy was successfully enforced or if a problem was encountered.

2.4.2.2.5 Database Components

The metapolicy database and the local policy database will be logically different, because they store semantically different policies, but theree will be a unique physical database that can be accessed by the components of the PBM system. Metapolicies are policies that will control the access to functionality and the behaviour of the PBM system, while policies control the access and behaviour of the active network resources managed by the management system.

2.4.2.2.5.1 Policies Database Components

Since meta-policy database and the local policy database are just logically different, they can provide the same interface to PBANEM as long as both of them use the same database technology. We expand upon some of the different database technologies that could be applied in FAIN.

2.4.2.2.5.1.1 RDBMS-based policy Database

If the Java programming language is used, JDBC can be used for the interface. If other languages or platforms is are used, ODBC can serve as the interface. Both of them focus on executing raw SQL statements and retrieving the results. We expect that higher-level APIs will be defined as well, and these will probably be implemented on top of this base level but not in this document. This document doesn't specify whether it is JDBC (Java DataBase Connectivity) or ODBC (Open DataBase Connectivity), focus is on the functionality of the interface needed by the PBM, which basically is for creating SQL statements which can add new policy to the database or retrieve the results (**Policy**) by executing those statements against relational databases.

2.4.2.2.5.1.2 Directory-based Database

Besides LDAP, Java Naming and Directory Interface (JNDI) can also provide applications written in the Java programming language with a unified interface to multiple naming and directory services, and more powerfully. JNDI enables seamless connectivity to heterogeneous enterprise naming and directory services. Developers can build powerful and portable directory-enabled applications using this industry standard. JNDI also supports LDAP v3.

2.4.2.2.5.2 Metapolicies Database Components

As mentioned previously, it is assumed that the metapolicies database and local policies database are physically unique and may have the same architecture.

2.4.2.3 Policy Representation

This section describes a proposal for the FAIN Policy Core Information Model and Policy Syntax. The proposal Syntax is based on the IETF PCIM extensions draft [30], with a few exceptions. In this draft a set of extensions are suggested to the PCIM model suggested in [17]. These extensions provide a model with a higher degree of flexibility for the definition of policy rules. On the other hand, this flexibility might, in some cases (e.g. PolicyRuleInPolicyRule aggregation) cause an increment in the complexity of the policy framework. However, this proposal allows for the level of flexibility and complexity that we want to give to the system, and eases the interoperation of the whole management system (e.g. through the definition of classes for variables and values). However, in our proposal we follow a mixed PCIM-PCIMe model for specifying actions (just one policyAction class), instead of following the PCIM extensions model (each policy action has to be specified with several classes depending on the number of variables that have to be changed). The reason for that decision is because we feel that the way in which policy actions are defined in the PCIM extensions draft might be unnecessary cumbersome (e.g. a quite simple policy action like the ipvpnPolicyEncryptionAction proposed needs 66 classes to be expressed with the PCIM extensions model). The FAIN Information model follows the simplePolicyAction and CompoundPolicyAction structure proposed in PCIMe but it does not have the PolicyVariableinPolicyAction and PolicyValueinPolicyAction classes, since variables and values are simply expressed as properties of the specific fainSimplePolicyAction subclass.

The policies themselves will be expressed in an XML document that will contain the necessary classes that describe the policy rule. This XML document will be carried by a Mobile Agent, which will carry other important information, e.g. security related information. Necessary parameters that need to be carried by the Mobile Agent are:

- Level of security: which indicates whether the XML document is either authenticated, encrypted, none or both of them.
- Security parameters: which allow the PBANEM system to authenticate and decrypt the received XML document.

Other parameters that might be interesting are:

- PDPId: Identifier of the PDP that has to process the policies described in the XML document.
- PDPUpdate: Boolean that when set to TRUE indicates that the PDP identified by the PDPId should be updated with a newer version before passing it the policies. This ability may provide some more flexibility to the system.

2.4.2.3.1 Advantages of XML for Policy Representation

Using XML as language for expressing policies has several advantages [34]. XML is ideal for transferring information between heterogeneous platforms because XML parsers are available for many platforms. Another advantage is that XML policy documents can be validated against an XML policy schema that resides on a remote, trusted server. This is possible because XML documents can carry a reference to their XML Schema, instead of the Schema itself. Moreover, the policy syntax checking functionality, is done intrinsically by the XML parser through the validation of the XML policy against its XML schema.

The mapping approach followed is based on using XML-Schema specification to define the vocabulary an XML policy will need to be validated against. All classes (e.g. PolicyVariable, PolicyValue, fainSimplePolicyCondition, fainSimplePolicyAction, etc. subclasses), properties, and even possible values, that conform to the FAIN Information Model, in all domains, would be expressed in one or several XML Schema classes. With this approach, we are able to specify and check the correct policy syntax of any possible fainPolicyRule, and even to check given property values are within some specific, valid range. As more details are given to the XML schema, more syntactic checks can be done to the policy by the XML parser.

One important issue about the mapping approach taken, is how the associations are done within the XML policy instances. In order to ease the readability and reduce the complexity and size of policy rules we map association classes as references to elements in the XML-Schema. This allows the policy to be more compact, simple and comprehensible than mapping all association classes.

Moreover, no information is lost, due to the use of this approach since all properties of aggregation classes are included in the reference as attributes. Therefore, all the information needed to translate the XML instance to objects (aggregation objects included) in a database is available.

The way, in which naming information is given, in an XML policy instance, is based on the proposal made in the CIM Core Policy Model specification [35]. That is, concrete properties of classes are used as keys for identifying instances. The only difference with the approach suggested in [80] for naming and the one adopted by FAIN, is that since we use references, instead of aggregation classes, there is no need to repeat the keys of the containing class in the contained class as suggested in [80].

The detailed specification of the initial FAIN management information model and its relation to the IETF model are given in appendix 8.1. We note that it is likely that this initial information model will undergo refinements and extensions as the work in FAIN progresses and particular requirements are discovered. Appendix 8.2 gives an example of how the mapping of the FAIN information model to an XML-Schema is done, and how the mapping of a fainPolicyRule, of the mapped information model, to an XML instance is done. To better understand these examples see [33], [36].

2.4.3 Physical Architecture of the System Components

Throughout this section details necessary for the physical implementation of the architecture proposed are given. These details are distributed amongf three main sub-sections. In the Communication Between Components sub-section (2.4.3.1) the internal interfaces between the main components of the architecture are described. The interface definition is given in IDL.The Interface Definition Language (IDL) has been chosen as language for the definition of the interfaces because of its huge popularity in Object Oriented technologies. Section 2.4.3.1.10, External Interaction Requirements, provides requireeriments to onrequirements to the interactions between the management system described in this document, and other FAIN systems that interact with it (e.g. the RCF interface of FAIN ANNs, the ASP system, and between the Network and the Element Level). Although the interactions between the Network and the Element Level are internal from the management point or view, they are included in this section because they are external from the "general" architecture presented point of view. Finally, in the appendix section but related with this chapter, the Description of Tools section, provides an overview of tools and technologies that could be used in the implementation of the architecture proposed.

2.4.3.1 Communication Between Components

2.4.3.1.1 Credential Check Interface- Int1

For an active network node (ANN), the ANN Manager must govern the ANN to restrict who can install active code or how much of the node resources they are allocated (via a policy) [60]. Additionally, the ANN Manager must also allow a packet to prove that the latter is a bona fide 'requestor' (a credential), and it must enable the ANN Manager to specify who (two best-known certificate systems are those of PGP and X.509) may issue such credentials (a trust relationship).

To describe the credential check interface, we note that each interface defines a new object type. Operation signatures are the essence of the interface, which are entry points for service request. As such the interface forms an opaque boundary between client code and object implementations. IDL declares what is exposed by this interface, and all other details are hidden.

Based on the sub-components given in Figure 26 and Figure 33, the credential check interfaces, in the form of IDL, are shown as *Credential.idl* and *Metapolicy.idl.*

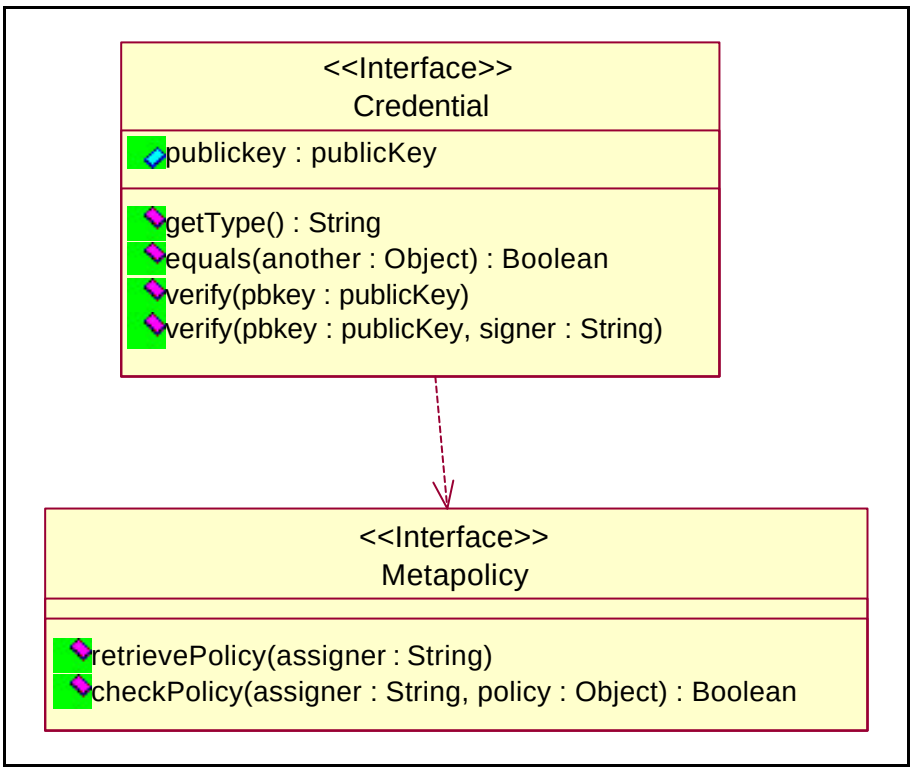

Figure 33 - Two interfaces used in credential check

Depending on the implementation, the Public key can be embedded in the credential or be stored in specific key base, or stored in policy repository with the meta-policy relating to the assigner of the credential. To make it more generic, we suppose that the public key is not embedded in the credential, therefore a public readonly attribute called *publickey* is defined. If public key is defined in the credential, an abstract method called getPublicKey can be defdined in interface Credential.

// Credential.idl

```
#ifndef __CREDENTIAL_DEFINED
#define __CREDENTIAL_DEFINED

module credentialCheck {
  interface Credential {
       // the public key for the corresponding actor
       readonly attribute publicKey publickey;

       // Returns the type of this credential.
       String getType();

       // Compares this certificate for equality with the specified object.
       Boolean equals (Object another);

       // Verifies that this credential was signed using the private key
       // that corresponds to the specified public key.
```

```
      // --- fulfil the functionality of credential check
      void verify(publicKey pbkey);

      // Verifies that this credential was signed using the private key
      // that corresponds to the specified public key, and the signer.
      void verify(string assigner, Object policy);
  };
};
#endif
```

// Metapolicy.idl

```
#ifndef __METAPOLICY_DEFINED
#define __METAPOLICY_DEFINED

module credentialCheck {
  interface Metapolicy {
      // retrieve meta-policy from the meta-policy database based on
      // the signer of the credential
      Object retrievePolicy (string assigner);

      // check if the entity (the signer of the credential) has the privilege
      // to do some element level management
      Boolean checkPolicy (string assigner, Object policy);
  };
};
#endif
```

The type of credential usually means the name of the algorithm used by the credential, which may include message digest algorithms (such as MD2,MD5), key and parameter algorithms (such as RSA) and digital signature algorithms (such as MD5withRSA).

2.4.3.1.2 Policy Conflict Check Interface – Int2

This conflict check interface should check for static conflicts derived from Policy Rules whose conditions are simultaneously satisfied, but whose actions conflict with those of currently existing rules . An important point to note is that rules may be 'time-based' (specifying an effective validity period in the future) or based on dynamic state information. These rules may indeed conflict with others. But, these conflicts may only be detected at the time that the rule becomes valid and enforcement actions are attempted.

The CDB interface contains a function that performs conflict evaluation on the submitted policy object. The function returns a result notifying the type of conflict –if any- found. Otherwise, the function returns normally.

```
module PDP {
  enum conflictType {MODALITY,
```

```
                    PRIORITY_FOR_RESOURCES,
                    DUTY,
                    INTERESTS,
                    MULTIPLE_MANAGERS,
                    SELF_MANAGEMENT};
  exception policyConflict {
    type: conflictType;
    resolved: boolean;       // true if the conflict was resolved.
  };                         // false otherwise.
  interface ConflictManager {
    void verify(inout policy: PolicyObject) raises policyConflict;
  };
};
```

The PolicyObject specifies the target of the policy. Therefore the CDB can use this information to locate other policies applied on the same target using the Database Access Module services.

2.4.3.1.3 PDP – Conflict Check Interface – Int3

The IDL interface would include a function to evaluate and process properly the policy, once checked.

```
module PDP {
  exception illegalExpression {
    string reason;
  };
  exception notFound {};
  interface PolicyEvaluation {
    void evaluate(in policy: sequence<octect>)
    raises illegalExpression;
};
};
```

2.4.3.1.4 Monitoring Service Interface – Int4

2.4.3.1.4.1 Introduction

Policy based management systems cope with controlling a large range of devices[10] by abstracting their common functionality and adapting the desired high level behaviour -specified by network operators- to their special features. However, monitoring has been traditionally achieved through *device* specific interfaces which limits the benefits that could be obtained from this technology.

[10] The term device is applied to the managed components either hardware or software.

This section discusses the development of a policy based monitoring system and describes the necessary interfaces.

2.4.3.1.4.2 Policy-based monitoring

Current systems do not consider flexible ways of controlling heterogeneous monitoring components but ad-hoc algorithms and interfaces suitable only for the tasks they have been designed for. Moreover, the monitoring information consumer is frequently hard attached to the metering blocks what complicates updating the monitoring mechanisms and makes it difficult to share the obtained information.

To resolve these two problems the monitoring system has been designed following a policy-based architecture. Monitoring policies will be used to control the monitoring components which play an enforcement point role. A decision point will be responsible for their proper operation, retrieving and deploying the applicable policies.

This way, monitoring domain PEPs become a source of synchronous and asynchronous messages for other domain PDPs. Since these monitoring data are to be available for every PDP, efficient distribution mechanisms have to be designed. For instance, fault management may require propagating an event raised in one domain which is a symptom of failure in another, possibly involving a policy decision. Mainly for this reason, an event channel has been though as the better alternative to disseminate information across different domains. It has also the advantage of decoupling consumers and sources what makes it simpler to add, remove or change the monitoring components – making the most of active network technology.

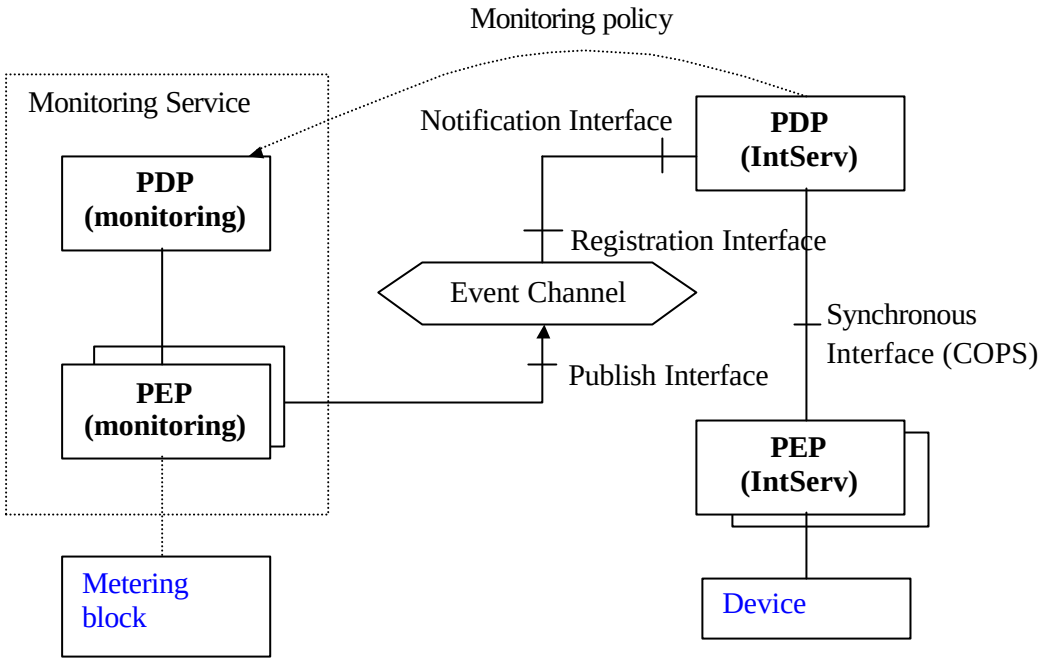

Figure 34 - Monitoring system and interfaces

Monitoring components are thus considered from two perspectives: as controlled elements they are policy managed; As sources of monitoring data they provide interfaces suitable for accessing them in an homogeneous way.

The proposed architecture complies the IETF standards requirements since only one PDP enforces policies on each PEP at a time. Monitoring policies[11] are delivered to the monitoring domain PDP just as application-specific policies and thus, following the same interaction pattern.

Monitoring information can also be retrieved by a PDP from the PEPs attached to it. However, note that in this case we are not dealing with monitoring specialised PEPs but with a part of the PEP normal working. The following sections describe these two modes of operation.

2.4.3.1.4.2.1 Monitoring Service Interfaces

Decision making processes often require knowledge, not only about the parameters appearing in a given policy, but also about the node or even the network state. Gathering this information may be attained either through a pull or a push model. The former one is related to the synchronous interface while the latter, often known as asynchronous interface, allows the notification of unexpected events. The operations supported by each of them is described following.

2.4.3.1.4.2.1.1 Synchronous interface

The synchronous interface allows the retrieval of monitoring data under demand. This can be accomplished either using COPS or additional pull interfaces.

In the first case, COPS DECISION messages will be used to ask subordinated PEPs for accounting data on the controlled devices. The Command-Code will be set to the *NULL* value meaning that no policy data is to be installed. Setting the Request-State flag to *true* will make the PEP to return a REPORT message containing monitored parameters in the format being specified in the policy information base defined for the domain.

[11] Note that for a not-monitoring domain PDP (for example, an IntServ PDP) monitoring policies express only the way it wants to receive service, not being the decision point for such policies.

Additional pull interfaces allow a PDP to ask for information to the monitoring PEPs. However it must be realised that in this interaction their behaviour does not correspond to decision/enforcement points and therefore it would have been inappropriate using COPS[12]. The CORBA Notification Service IDL interfaces fulfil all the requirements imposed to the event channel. Adapting these standard interfaces to the PBNM system characteristics assures compatibility with existing event channel implementations (also improving the interoperability of the overall system).

The required operations are defined as follows in [65][13]:

```
module CosNotifyComm {
        //…
        interface StructuredPullSupplier {
                StructuredEvent pull_structured_event() raises (Disconnected);
                //…
        }
}
```

A monitoring domain PEP may require several pull-suppliers, each implementing the StructuredPullSupplier interface, in order to allow the synchronous retrieval of different types of information. The pull_structured_event method will be used to obtain the required data (which will be included in an StructuredEvent) from a proxy supplier provided by the channel. As a consequence the behaviour of this method will be slightly different from the specified in [65], since the event becomes just a means to pass the monitored data from the PEP to the PDP loosing its asynchronous facet.

2.4.3.1.4.2.1.2 Asynchronous interface

The monitoring system can also provide information in an asynchronous way by notifying event occurrences. The CORBA Notification Service interfaces are suitable for this purpose. Next sections analyse how the functionalities and interfaces described in the standard can be used in the PBNM system.

2.4.3.1.4.2.1.2.1 Registration interface

After obtaining a proxy supplier from the event channel, the PDP is allowed to subscribe to the events it is interested in. Getting the proxy supplier involves two steps: first, the PDP gets a consumer admin object from the channel and then the operation obtain_notification_push_supplier is invoked on that object to request an structured event supplier. Finally, the PDP connects itself to the channel using the connect_structured_push_consumer method found in the StructuredProxyPushSupplier interface.

```
interface EventChannel {
        //…
        ConsumerAdmin get_consumeradmin(in AdminID id) raises(AdiminNotFound);
}
interface ConsumerAdmin {
```

[12] Furthermore, COPS does not support the interaction of a PEP with several PDPs using the same client-type.

[13] For clarity reasons, the inheritance references have been removed. The method definitions are only intended to be informative. Please refer to the CORBA Notification Service for a complete description of the interfaces.

```
        //…
        ProxySupplier obtain_notification_push_supplier( in ClientType ctype, out ProxyID proxy_id)
          raises (AdminLimitExceeded);
        // The PDP should set the ctype field to the value STRUCTURED_EVENT
        //…
}
interface StructuredProxyPushSupplier {
        void connect_structured_push_consumer( in StructuredPushConsumer push_consumer)
                raises (AlreadyConnected);
}
```

Therefore, the PDP must implement the StructuredPushConsumer interface which includes one function to disconnect the PDP from the channel.

```
interface StructuredPushConsumer {
        //…
        void disconnect_structured_push_consumer();
}
```

The PDPs have then the possibility to know all the available event types, i.e. the events that could be raised by the PEPs. The list is obtained by calling the method obtain_offered_types on the ProxySupplier interface.

```
interface ProxySupplier {
        //…
        EventTypeSeq obtain_offered_types(in ObtainInfoMode mode);
}
```

The CORBA notification service implements event subscription through filter objects. The PDP informs to the channel about the events it wants to receive by configuring a filter and attaching it to the proxy supplier. Only events matching the filter rules will be forwarded to the PDP.

Filter configuration will be realised by the PDP depending on the conditions appearing in the policies whose deployment it is responsible for. The FilterFactory and Filter interfaces [65] will allow specifying the structured events to be filtered.

Once the filter has been set up, the PDP attaches it to the proxy supplier by calling the add_filter operation on it. PEPs are not involved in any of these procedures and do not have to implement any filtering related mechanisms.

2.4.3.1.4.2.1.2.2 Publication interface

Publishing events is a means to inform to the PDPs about the available monitoring information. The PEP will be connected to the event channel by means of a proxy consumer. This object is obtained following the same steps as described formerly. After getting a supplier admin reference from the event channel, the PEP must invoke the method obtain_notification_push_consumer. Calling connect_structured_push_supplier on the returned object will connect the PEP to the channel.

```
interface EventChannel {
        //…
        SupplierAdmin get_supplieradmin(in AdminID id) raises (AdminNotFound);
        //…
}
interface SupplierAdmin {
        //...
        ProxyConsumer obtain_notification_push_consumer(
                in ClientType ctype,
                out ProxyID proxy_id) raises (AdminLimitExceeded);
        //...
}
interface StructuredProxyPushConsumer {
        //...
        void connect_structured_push_supplier(in StructuredPushSupplier pushSupplier)
                raises (AlreadyConnected, TypeError);
        //...
}
```

The PEP must then announce the event types (and relevant information) it is prepared to deliver. Since the proxy consumer returned by the channel implements the NotifyPublish interface, it is enough to call the method offer_change on that object.

```
interface NotifyPublish  {
        void offer_change (EventTypeSeq added, EventTypeSeq removed)
                raises (InvalidEventType)
}
```

Publishing an event does not imply it will be automatically delivered. The notification service offers services that allow the PEP to know whether there is any PDP interested in the events it is capable to produce. Only in such a case the PEP will actually generate the events.

2.4.3.1.5 PDP - Monitoring Service Interface – Int5

2.4.3.1.5.1 Notification interface

The notification interface allows the delivering of events to PDPs through the channel. In fact, the event will move through several objects, beginning from the PEP to the proxy consumer and, after dissemination through the event channel, to the proxy supplier reaching finally the PDP. However, fortunately this process is mainly hidden.

The only requirement for the PDP to be able to receive events is to implement the StructuredPushConsumer interface. This interface contains the method push_structured_event, that will be called directly by the event channel.

```
interface StructuredPushConsumer {
        void push_structured_event (in StructuredEvent notification)
                raises (Disconnected);
        //…
}
```

2.4.3.1.5.1.1 Event Format

The CORBA Notification Service provides what is called structured events. While not being untyped events, they enable to map a wide variety of event formats. In our case, it would be desirable the event fields to correspond to the defined in the policy information base for the domain.

Mapping PIB events to the structured event format is still to be defined for each management domain.

2.4.3.1.6 PDP – PEP Interface

In the PDP-PEP interaction two main scenarios can be distinguished depending on their distribution among the network nodes. In the first case, the PDP and the PEP appear in different network nodes whereas in the second one both components are integrated into the same node. Selecting any of the previous configurations will depend on the functionality that is to be provided, being both likely to appear in the FAIN project. Within the IETF framework, the defined PBNM architecture proposes several solutions to implement PDP-PEP communication depending on the considered scenario. Nevertheless, it is our aim to develop a framework providing a high degree of location transparency, accessing the same interfaces both for local and remote interaction.

In order to support the extension of the policy information model as a means to include new types of policies when necessary, we have based the interface definition in the COPS-PR proposals. This model also allows the PEPs to dynamically bind to the existent PDPs, which becomes essential if dynamic instalation of new components –i.e. elastic management- is to be provided.

The interface definition comprises the policy server interface and the policy client interface definitions. The PDP should implement the policy server interface whereas the PEP should provide the policy client declared methods.

2.4.3.1.7 PEP Interface – Int6

The PEP and the PDP components have to interact, in order to evaluate and successfully enforce the system policies. The COPS-PR protocol, which has been defined by the IETF RAP workgroup, can be used for the communication between those two components.

COPS-PR allows for the establishment of a session between a PEP and a PDP, the reporting of PEP status information to the PDP, the request of a policy decision from the PEP, the download of policy decisions from the PDP to the PEP.

The operations that need to be supported by the PEP interface are the following:

```
interface PolicyClient {
```

void decision(in ClientType clientType,

in Handle clientHandle,

in PolicyDecision decision,

in boolean solicited)

This operation is used by the PDP to download the policy decision to the PEP. The parameters of this operation are a client handle, which identifies the request state of the PEP , the policy decision, and the solicited flag, which indicates if the decision is sent as provisioning and not after a request from the PEP.

The decision() method can be called synchronously by the PDP in response to a previous request issued to the policy server interface (outsourcing scenario), or asynchronously to provision policy data (provisioning scenario) either because of receiving a new policy or as a consequence of detecting a system state change. The solicited flag distinguishes each of these scenarios.

void synchroniseState(in ClientType clientType,

in Handle clientHandle)

With this operation the PDP requests from the PEP to update its state

void clientAccept(in ClientType clientType,

in Timer KA,

in Timer ACCT)

The communication between the PEP and the PDP is initiated by the PEP, using a Client-Open message. The PDP responds with Client-Accept message to confirm the establishment of the session. The two timer parameters define the maximum intervals for the keepAlive messages and accounting state updates sent by the PEP.

```
void clientClose(in ClientType clientType,
                 in Error errorInfo)
```

The PDP may send a Client-Close message, to inform the PEP that it cannot be supported any longer. This message may also come as a negative response by the PDP to a Client-Open message previously sent by the PEP. The Error parameter specifies the reason that caused the end of the communication.

void keepAlive(in ClientType clientType)

}

The PEP has to send KeepAlive messages to the PDP, after specified intervals. When the PDP receives such a message, it responds by sending a KeepAlive back to the PEP. This mechanism validates that both sides of the communication are still functioning, when no other messages have been exchanged for a time period.

2.4.3.1.8 PDP – PEP Interface – Int7

In this document an outline of the communication interface between the Policy Decision Point and the Policy Enforcement Point is provided. The interaction based on the methods defined in the interfaces is depicted in sequence diagrams.

In the PDP-PEP interaction two main scenarios can be distinguished depending on their distribution among the network nodes. In the first case, the PDP and the PEP appear in different network nodes whereas in the second one both components are integrated into the same node. Selecting any of the previous configurations will depend on the functionality that is to be provided, being both likely to appear in the FAIN project. Within the IETF framework, the defined PBNM architecture proposes several solutions to implement PDP-PEP communication depending on the considered scenario. Nevertheless, it is our aim to develop a framework providing a high degree of location transparency, accessing the same interfaces both for local and remote interaction.

In order to support the extension of the policy information model as a means to include new types of policies when necessary, we have based the interface definition in the COPS-PR proposals. This model also allows the PEPs to dynamically bind to the existent PDPs, which becomes essential if dynamic instalation of new components –i.e. elastic management- is to be provided.

The interface definition comprises the policy server interface and the policy client interface definitions. The PDP should implement the policy server interface whereas the PEP should provide the policy client declared methods.

2.4.3.1.8.1 Policy Server Interface

The policy server interface contains session management functions as well as policy information exchange methods. The session concept is a means to identify concurrent interactions which relate to different client types –as defined in COPS-PR– implemented by the same policy enforcement point.

A policy client initiates a session by issuing an `open()` call. The parameters identify the client type and provide a unique identifier that designates the PEP within a domain. Additional information about the policy types the PEP supports, default policy configuration, etc, can be included in the client specific information field[14]. All these data is registered locally by the PDP.

The policy server can either accept or reject the open request. Rejection is explicitly signalled by raising the `rejected` exception. If the function returns normally, the open request has been implicitly accepted. The `rejected` exception provides an error code pointing out the reason.

```
typedef unsigned short ClientType;
typedef sequence<octet> ClientSI;
typedef sequence<ClientSI> ClientSIList;
struct Error {
     unsigned short ErrorCode;
```

[14] The client specific information format depends on the client type and should be provided as an interface definition extension.

```
      unsigned short ErrorSubcode; };

exception rejected {
      Error reason; };

interface PolicyServer {
      // If not rejected, it has been implicitly accepted.
      void open( in clientType ClientType,
                 in PEPIdentifier string,
                 in clientInformation ClientSIList )
      raises (rejected);
      void close(in clientType ClientType,
                 in reason Error );
      …
};
```

The `close()` method is invoked by the policy client to inform that the specified client type is no longer supported. As a consequence, the session for that client type is closed. Additional information about the reason is provided.

The methods `request()` and `report()` relate to policy distribution. The method `request()` is invoked by the policy client when it is unable to manage a detected configuration change with the previously provisioned policies. The field `requestType` identifies the kind of request, while the `clientHandle` is an opaque identifier used by the PDP solely to keep track of the request state.

The PEP should call `report()` in response to a decision coming from the PDP. The `reportType` informs whether the directives have been successfuly followed or not. The `report()` method can also be asynchronously called to provide monitoring information

Finally, the method deleteRequest() has been added to allow the client to remove a request state kept in the server.

```
void request(   in clientType ClientType,
                in clientHandle Handle,
                in requestType RequestType,
                in clientInformation ClientSIList )
raises (rejected);

void report(    in clientType ClientType,
                in clientHandle Handle,
                in reportType ReportType,
                in clientInformation ClientSIList );
```



```
void deleteRequest(   in clientType ClientType,
                      in clientHandle Handle ); //reason?
```

2.4.3.1.8.2 Policy Client Interface

The policy client interface contains the `close()` method that allows the PDP to close a session for a given client type. Therefore, a session can be closed either by the PDP or the PEP.

The `decision()` method can be called synchronously by the PDP in response to a previous request issued to the policy server interface (outsourcing scenario), or asynchronously to provision policy data (provisioning scenario) either because of receiving a new policy from the Network Management Node or as a consecuence of detecting a system state change. The `solicited` flag distinguishes each of these scenarios.

Both methods have been further described in section 0.

2.4.3.1.8.3 Complete IDL definition of the interfaces

Complete IDL definition of the Policy Server and Policy Client interfaces.

```
module PBNM {
     typedef unsigned short ClientType;

     // Several request types could be defined depending on
     // requirements
     typedef enum {CONFIGURATION}    RequestType;
     typedef enum {DECISION, ERROR} ObjectType;
     typedef enum {NULL, INSTALL, REMOVE} CommandCode;
     typedef enum {SUCCESS, FAILURE, ACCOUNTING} ReportType;

     // The actual PolicyInformation structure MUST be given for
each
     // client type.
     typedef sequence<octet> PolicyInformation
     typedef sequence<octet> Handle;
     typedef sequence<octet> ClientSI;

     struct Decision {
               CommandCode command;
               unsigned short flags;
               PolicyInformation policyInformation; };

     typedef sequence<ClientSI> ClientSIList;
     typedef sequence<Decision> DecisionList;
```

```
struct Error {
          unsigned short ErrorCode;
          unsigned short ErrorSubcode; };

exception rejected {
          Error reason; };

interface PolicyServer {
     // If not rejected, it has been implicitly accepted.
     void open( in clientType ClientType,
                in PEPIdentifier string,
                in clientInformation ClientSIList )
     raises (rejected);

     void close(     in clientType ClientType,
                in reason Error );

     void request(   in clientType ClientType,
                     in clientHandle Handle,
                     in requestType RequestType,
                     in clientInformation ClientSIList )
     raises (rejected);

     void report(    in clientType ClientType,
                     in clientHandle Handle,
                     in reportType ReportType,
                     in clientInformation ClientSIList );

     void deleteRequest(  in clientType ClientType,
                          in clientHandle Handle ); //reason?
};
```

2.4.3.1.9 Policy Database Interface – Int9

Policy database access interface provides basic functions such as retreive and store data. In addition from view point of authorisation each component might need to make connection with policy database component before they access policy data. The definition with IDL below includes only basic functional interfaces.

```
interface AccessPolicyDB{
        PolicyDB  open_policydb(in policydb_server ); //AN operation to connect with policyDB
server
      void close_policydb(in policydb);   //AN operation to disconnect with policy database server
      get_policyobject(in policy_id);              //An operation to retrieve policy data object
        set_policyobject(in policy_id);            //An operation to store policy data object
        delete_policyobject(in policy_id);         //An operation to delete policy data object
}
```

In the following we just show two most important interfaces: PolicyContext and PolicyDirContext.

PolicyContext is the core interface that specifies a policy naming context. It defines basic operations such as adding/deleting a policy, adding a name-to-object binding, looking up the object bound to a specified policy name, listing the bindings, removing a name-to-object binding, creating and destroying sub-policy etc. [56]

The PolicyDirContext interface enables the directory capability by defining methods for examining and updating attributes associated with a policy.

```
public interface PolicyContext {
   public void addPolicy(Name fainPolicyName) throws NamingException;
   public void deletePolicy(Name fainPolicyName) throws NamingException;
   public Object lookup(Name fainPolicyName) throws NamingException;
   public void bind(Name fainPolicyName, Object obj) throws NamingException;
   public void rebind(Name fainPolicyName, Object obj) throws NamingException;
   public void unbind(Name fainPolicyName) throws NamingException;
   public void rename(Name fainPolicyName_old, Name fainPolicyName_new) throws
NamingException;
      ...
   public Context createSubpolicy(Name fainPolicyName) throws NamingException;
   public void destroySubpolicy(Name fainPolicyName) throws NamingException;
   ...
};

public interface PolicyDirContext extends PolicyContext {
   public Attributes getAttributes(Name fainPolicyName) throws NamingException;
   public Attributes getAttributes(Name fainPolicyName, String[] attrIds) throws NamingException;
   ...
   public void modifyAttributes(Name fainPolicyName, int modOp, Attributes attrs)
       throws NamingException;
   public void modifyAttributes(Name fainPolicyName, ModificationItem[] mods)
```



```
        throws NamingException;
    ...
}
```

2.4.3.1.10 External Interaction Requirements

2.4.3.1.10.1 Interaction Requirements Between the Network and the Element Level (Int8)

The interaction requirements between the Policy-based Active Network Management (PBANM) system and the Policy-based Active Network Element Management (PBANEM) system can be easily split in two main groups of requirements. Those needed to allow the network level to send policies to the element level, and those needed to allow the network level to receive information from the network level.

In the next two section, we will describe in detail how these interactions are done, and which requirements are necessary to achieve interoperability between both management levels.

2.4.3.1.10.2 From Network to Element Level

The network level will send his orders to the element level in the form of policies. Therefore, we can clearly obtain two main requirements from here in order to achieve interoperation, these are, have a common XML syntax of the policies, and agree in a way of transporting these policies.

The first requirement, common XML syntax, can be further divided in two sub-requirements, have a common knowledge of the FAIN Information Model, and a common XML-Schema representing this Information Model. The information model, will determine which policy classes, e.g. conditions, actions, can be sent, while the XML-schema determines how an XML policy will represent the information of these policy classes.

Both, the FAIN Information Model, and the XML-Schema, can be found within this document, therefore, interoperation is achieved regarding the XML policy syntax.

However, to achieve complete interoperability from the network level to the element level we still have to find a common way of transport. Several possibilities can be found for carrying policies between different levels: mobile agents, active packets, XML-RPC [40], SOAPRPC [39]. In FAIN, the mobile agents approach will be followed. The mobile agents approach has several advantages against the others, some of these are:

- The mobile agent carries additional information regarding the policy as a whole. For example, whether the policy is encrypted, authenticated, both, none, the security keys, information about the PDP that has to process the policy, etc.
- One mobile agent can carry several policies and apply them atomically or in several PBANEM nodes.
- The network level does not need to remotely access the element level interface, it just sends the mobile agent and waits for an enforcement result (if needed).

Using mobile agents as transport of policies solves many interoperation issues as well. Basically, the only requirement is that both levels make use of the same mobile agents platform.

2.4.3.1.10.3 From Element to Network Level

The Network Management Level will receive all the management information from the element level in the form of event reports. Moreover, the network level is able to determine which events it wants to receive (by setting the appropriate policies within the PBANEM). For example: the network level sends a policy that indicates to the element manager that it should send to the network manager events every time a policy is correctly enforced (or when the element manager can not enforce the policy for any reason). Also, it can send a policy that determines that the element manager should send an event when an specific fault occurs, then when the element manager receives that event it might react realising the correspondent configuration changes.

Following this approach implies that we have one main interoperation requirement, that is, specify a common syntax for event reports and its mapping to XML and XML-Schema as we have done with policies. In the appendix (see R12-Appendix), an example of event report in XML and XML-schema is given.

The structure of classes that represent the types of events that can be send between the element level and the network level are described below. These are all abstract classes from where concrete, instanceable classes have to derive from extending the properties of the abstract classes to its necessities. As an example, an instanceable class is given as well.

```
CLASS fainEvent
Description:""
Abstract=True
Properties:
        CreationClassName       //String that identifies the name of the class of the instance
        EventName       //String that identifies the instance of the event class
        ResourceIdentifier      // list of strings identifying the resource that sent the event
        CorrelatedEvents        //list of strings that identify events that are somehow related with this one

CLASS fainFaultEvent:fainEvent
Description:""
Abstract=True
Properties:
        probableCause           //String that identifies the probable situation that caused the event
        perceivedSeverity        //string that identifies the severity of the fault
        affectedResources       //list of identifiers of resources affected by the fault

CLASS fainRCReportEvent:fainEvent
Description:""
Abstract=True
Properties:
        resourceComsumption   //This property carries a list of values regarding the consumption of resources per user

CLASS fainMonitoringEvent:fainEvent
Description:""
Abstract=True
Properties:
```

```
        monitoredValue      //list of properties-value pairs monitored from the element level


//Example of instanceable event class.
CLASS fainServiceDownEvent:fainFaultEvent
Description:""
Abstract=False
Properties:
        relatedServiceComponents      //list of strings identifying components that interact with this one to offer a service
        AssociatedVE      //list of identifiers that determine the Virtual Environment where that service run
        AdditionalInfo      //string with additional information regarding with the fault occurred.
```

2.4.3.1.10.4 Interaction Requirements Between the NMS and the ASP (Int11)

The Network Management System (NMS) needs to interact with the ASP system under three possible situations, to upgrade the management system itself, when a service is required and has to be installed in the active network, and when an active packet needs some active code running in an ANN to be processed correctly and request the downloading, or installation, of this code.

In the first case, the NMS will ask the ASP to download the necessary blocks in the appropriate interface that may be pointed by the management system. The interaction requirements imposed by this situation are quite straightforward. The ASP needs to offer an interface to the management system where a component download can be requested. We may give, as optional parameters, a pointer to the "location" where the code has to be downloaded to, and a reference to an specific code server where this code can be found.

The second situation where there will be an interaction between the ASP and the NMS is when a policy of the kind: "if flow=X then treat it with service Y" has to be processed by the management system. In this case, the NMS should reserve the computing (and probably bandwidth as well) resources needed by serviceY and flow X respectively. The network management system will need to interact with the ASP in order to determine which code modules should be installed, where, and which resources should be reserved to each module.

In order to obtain this information, the NMS will need to access an ASP interface. It will pass to the ASP the identification of the service, and optionally, the resources it has requested (this information can be useful to the ASP in order to provide one code version more powerful but more resource consumer as well, or another code version much lighter) and the type of VE that are available. The ASP will answer to that request with the actual code modules that should be installed, the type of ANN (or VE) where they should run, and possible dependencies between them and other modules. Optionally the ASP can provide as well, information about the minimum resources that these code modules might need. The NMS will use this information to find the appropriate nodes where the code should be installed.

Once the resources and the modules have been determined, the NMS will reserve the appropriate resources (if there are enough resources and no conflicts with other policies). If the reservation of resources have succeeded, the NMS will contact the ASP to request the installation of the service in the node. In order to be able to make such a request the ASP has to offer an interface to the NMS where active code can be requested to be downloaded in an active network node. The NMS will need to supply a pointer and a credential, in order to allow the ASP to install the service in the correct place (pointer), and to allow the services to access their assigned resources.

The third possible scenario where the NMS and the ASP system might interact is when a packet points to (or carries) an active code to be processed with[15]. In this case, the active service provisioning facilities within the Active Network Nodes will send a request to the NMS asking if the code can be installed with the requested resources or not. The NMS will then make a decision and answer and realise the appropriate configuration actions according to that decision (i.e. if the code is allowed to be installed the appropriate resources have to be reserved to that code). Once the resources are reserved the NMS will answer positively to the service provisioning facilities of the ANN, which will download and install the active code to process the packet.

This third scenario implies, basically, two interactions between the NMS and the ASP: the request and the response. The NMS has to offer to the ASP an interface where the ASP can request for a decision about the installation or not of some code with its assigned resources. The ASP has to provide information about the code, the resources requested, and a credential of the user who sent the code, in order to allow the NMS to make a decision.

The response from the NMS to the ASP, has to give information regarding the decision taken (yes/no) and if the response is positive, it needs to provide as well, a pointer to where this active code has to be installed.

2.4.3.1.10.5 IDL- like required methods description

This interface will be called by the NMS in the first situation (to upgrade the NMS itself).

```
interface ASP{
        void downloadReq(in:CodeID string, in:Credential Credential, in:Pointer string);
}
```

The parameters that would be required for this method should carry information related with:

CodeID, which will be a string identifying the needed code package.

Credential: information regarding the principal associated with the principal who made the request.

Pointer: is a string that just indicates to the ASP the place (e.g. a directory) where the package should be downloaded.

For the second situation, the required method will be slightly different.

```
interface ASP{
        void serviceMap(in:ServiceID string, in:Credential Credential, in:ReqResources ResourceList, out:CodeModulesInfo CodeInfoList);
}
```

The parameters that would be required for this method should carry information related with:

[15] Only when the active code requested will be used to process a flow of packets. In the capsule approach it is not feasible, yet desirable, to be constantly requesting the NMS for a decision.

ServiceID, which will be a string identifying the service about which we are requesting information.

Credential: information regarding the principal associated with the principal who made the request.

ReqResources: this parameter will carry information about the resources requested to that service.

CodeModulesInfo: this parameter will be a list of CodeInfo structures. Each of these structures will carry information regarding: the code module that should be installed, the type of ANN and the VE where it should be installed, dependencies with other code modules and optionally the minimum recommended resources for that code module.

In the third situation, we have to think in the method offered by the NMS to the service provisioning facilities:

interface NMS{

boolean installACReq(in: CodeId string, in:Credential Credential, in: Resources ResourceList, out: Pointer)

}

The method returns a boolean that tells if the reservation has been done or not.

The parameters of the method are:

CodeID, which will be a string identifying the needed code package.

Credential: information regarding the principal associated with the principal who made the request.

ResourceList: will be a structure reflecting the computing resources requested.

Pointer: A pointer to where the code should be installed.

2.4.3.1.11 Interaction Requirements Between the RCF and the NMS (Int10)

2.4.3.1.11.1 Introduction

The scope of this document is to provide the functional requirements from the PBANEM framework to the RCF and partially to identify their interrelation. This is crucial because it will establish up to a great extent the interface needed between the PBANEM and the RCF. One of the main goals of active networking is to incorporate QoS-aware applications. This includes certain steps, such as admission control, resource reservation, differentiated class of service, packet marking, congestion avoidance, priority-based queuing.

In the remainder of this document, we will try to address most of these issues, from the scope of the interface requirements.

2.4.3.1.11.2 Requirements

2.4.3.1.11.2.1 Management/Rcf Interface for accounting

The management framework uses high-level policies that describe certain actions, such as resource allocation, access control etc. These policies are translated into lower level instructions by the Policy decision point (PDP), which feeds one or many Policy Enforcement Points (PEPs). In other words, inside the PDP the policy is translated into a set of commands that is understandable by the appropriate PEP. The PEP communicates with the physical or logical resources of the active node through the RCF interface. This interface will be described in this document as the Management-RCF Interface (also see Figure 35-by courtesy of GMD).

In RCF there is a hierarchical categorisation of resources. Resources are firstly allocated per VE, then these resources are split to ones allocated per EE. There is also the notion of resource allocation per application, or per user.

The above implies that the granularity of **resource monitoring** that is responded from the RCF back to the element management system differs. There should be responses per VE, per EE, per application, or per user. This also means that the categories of **resource accounting** (based on IDs) inside the element database must be per VE, per EE, and per application/user.

2.4.3.1.11.2.2 Fault Management

The Management/RCF interface must include methods that help to act against unexpected, troublesome situations.

When an active node breaks down for example, (the active node is disconnected from the management system), the RCF Resource manager should store the configuration status of the active node, and then it must generate a **recovery report** to the PBANEM system when the connection is restored. A necessity for the RCF component is also to make **periodical backups** of the resource allocation table, in case the RCF itself brakes down.

When the monitoring facility of the RCF realises that a particular portion of code misbehaves, the monitoring mechanism of the PBANEM system should be notified immediately. Thus, a significant requirement is that the RCF component(s) (Resource manager or policy targets), must be able to **send events asynchronously** to the PEP. The PEP then informs the PEP manager of the PDP, and the appropriate fault management policy is transferred from the database and applied to the faulty node.

For example, a policy might dictate that if the difference between the agreed level of resource usage according to the SLA and the actual one is big (a threshold needs to be set), the RCF framework is instructed to **stop the code** on the fly. If the difference between the agreed level of resource usage and the one being used is not very big, the user on behalf of whom the code is running, could receive a warning from the PBANEM system. As a note, in management parlance we regard the policy targets identical to the logical-physical resources.

Moreover, the PBANEM must be able to **specify** to the RCF the situations (conditions) that cause an event to be sent (from the RCF to the PBANEM).

The picture below shows how management is performed in all three levels (Network management level, network element level, active node level).

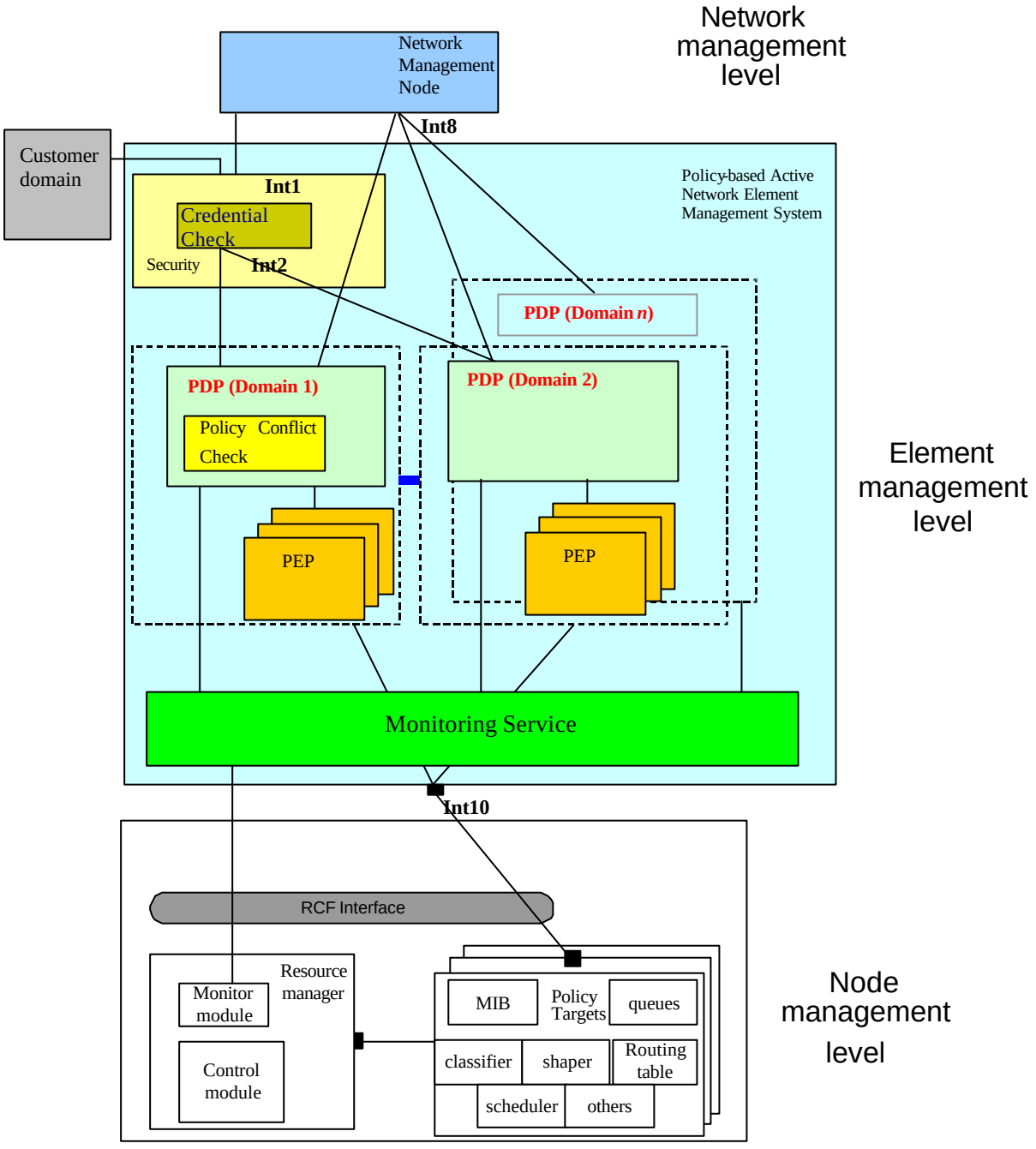

Figure 35 - Phase one scenario

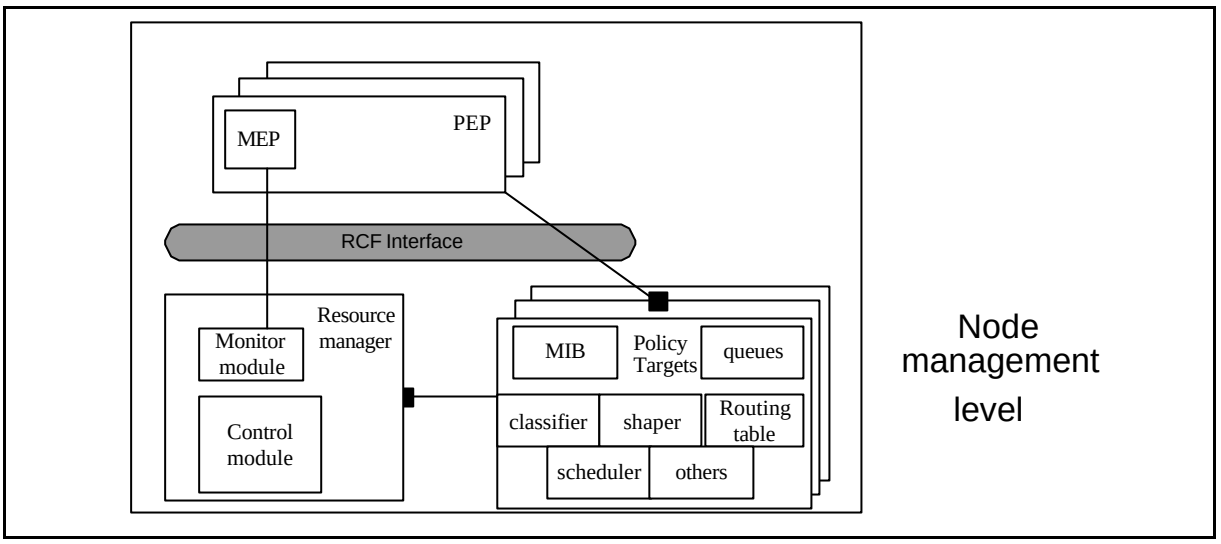

Figure 36 - Phase 2 scenario

In the second version of the management framework, the node management level may be similar to the one shown in Figure 36. What has basically changed from the first version, is that the PEP now resides at the node management level. Possibly, in an attempt to make the node even smarter, some basic PDP components could reside at the node level as well.

2.4.3.1.11.2.3 Configuration management

The configuration management functionality could be carried out in three different ways:

Provisioning, signalling and self-adaptation.

In the *provisioning* scenario, either the customer or the network manager, send policies to the network element in order to configure the active node's resources. So the PBANEM system must support the necessary mechanisms for QoS provisioning: IntServ, Diffserv. In the *signalling* scenario, the policy target receives a reservation request, (an RSVP packet) and passes the request to the element manager. In the *self-adaptation* scenario, it is the PBANEM system itself that reconfigures the resources, after detecting a change on the network status.

Considering the above we can conclude that the RCF system should (when suggested by policies of the PBANEM) be able to **identify different flows** based on certain fields of the packet header such as the source/destination IP addresses, the protocol used, or the source/destination ports).

Moreover:

- The Interface between the PBANEM system and the RCF must allow the **control of the complete lifecycle** of both the VEs and the EEs. This means that the PBANEM framework is responsible for creating, modifying, deleting VEs/EEs.
- The RCF system is responsible for the **resource allocation to the VEs and the EEs.**

  By resources we mean both computational and communicational, physical and logical.

  Thus the RCF should allow the **reservation of bandwidth resources to flows**, as well as the **assignment of flows to EEs.** The RCF is also responsible for the **access and the modification of the routing tables** on a per flow granularity.
- The RCF framework must be able to perform **priority based/separate queuing**, for performance critical applications.

- The **installing and removal of code within the VE** can be thought of being a ASP issue, so the interface must allow these actions as well.
- The interface should be **the same for every VE.**
- When the authentication engine gives the OK, the PBANEM system after checking the PBANEM database, can **dictate the level of NodeOS functionality** each VE can have access to. After that is done, the VE does not have to interact with the PBANEM system on that matter again, until the session is over.

2.4.3.1.11.2.4 Performance management

The active node should offer information about the **current status of resources** inside the node.

It should provide information about the status of the computational resources, both in total and per flow. For example this includes the CPU use percentage, the used/free memory, used/free storage space, the number of threads in the node, the number of threads per execution environment etc.

Also, information is needed about the communication resources and the forwarding status. This includes, used/free bandwidth, queue status, dropped packets, per flow of packets.

The performance management can in that sense be very helpful in preventing congestions(congestion management).

2.4.3.1.11.2.5 Security management

When a resource reservation is requested to the management system, the management/RCF interface must provide **authentication information** of the requesting principal to the management system.

This is the so-called access control, and should be performed by the security framework which needs to reside both in the PBANEM system (user-access management), as well as in the active node. (If these are implemented in different physical nodes of course.)

2.4.3.1.11.3 Scenarios

The understanding of the whole procedure will be better through scenarios. Lets take for example a signalling and an EE-creation scenario.

2.4.3.1.11.3.1 Signalling scenario

The goal of this scenario is to show how the node can reserve resources, say bandwidth, requested by a signalling packet (i.e. an RSVP packet).

1.- An RSVP packet arrives at the active node(policy target). The signalling-reservation capable policy target sends the query up to the appropriate PEP along with a credential of the actor who wants to reserve the resources. This credential should encapsulate the ID of the user who wants to reserve resources. As a reminder, all users and all resources are represented by a well-known ID, which are all stored inside the PBANEM database.

2.- The PDP will check if there is any policy in the database allowing that customer to reserve node resources, based on the ID that the user possesses.

2.1.- If the customer is not allowed to reserve node resources an error message is sent back to the PEP that sent the query. The PEP informs the node resource manager not to grand any resources for that particular ID.

3. - If the customer is allowed to grand resources, the PDP will then ask the PBANEM database about the resources already assigned to that customer. The PBANEM monitoring mechanism will also ask the RCF monitor module, which resides at the node level about the actual resource consumption of the user.

3.1.- If the requested resources plus the resources already assigned/consumed are higher than the maximum resources allowed, an error message is sent back to the PEP. (Thus both the Monitoring mechanism of the RCF framework and the PBANEM database need to be consulted).

4.- The PDP tells the PEP to realise the correspondent reservation actions.

5.- Finally, the PDP informs the network manager of the actions taken in the resources of the node due to the successful processing of the request.

2.4.3.1.11.3.2 EE creation scenario

The EE creation scenario is part of the configuration management and it is shown below:

1) A user requests the creation of an EE.
2) The PBANEM system checks existing policies to see if the user who has made the request has the necessary privileges and if there are sufficient resources in the node for this EE. This means that the functionality of the privileged EE, who had the task of controlling the lifecycle of all EEs, is now performed by the PBANEM system.
3) If the user has no privileges whatsoever to install an EE, an error message is send back.
4) If the user has the necessary privileges, but there are not enough resources inside the node, then an alternative node can be proposed by the PBANEM.

If the user has the necessary privileges and there are available resources, the EE is created.

2.5 R12 APPLYING THE ARCHITECTURE TO ELEMENT AND NETWORK LEVEL

2.5.1 Specific issues for the architecture at element level

2.5.1.1 Introduction

The purpose of this document is to remind ourselves of the PBANEM basic functionality and to suggest some additional functionality that will possibly be needed in the future.

2.5.1.2 Functionality of the PBANEM

PDP: Receives policies from the customer or the NIP and translates them into a PEP-understandable format.

PEP: Receives policies from the PDP and transfers them to the policy targets.

Policy Targets: Physical and/or logical resources that reside at the node level.

Monitoring Service: Receives conditions from the PDP and monitors them in order to become true. When they do, it notifies the PDP. The monitoring service is in close co-operation with the monitoring module of the RCF.

Database: Stores policies, users credentials, etc.

2.5.1.2.1 Configuration management

The configuration management functionality needed is:

- QoS-Provisioning: Either the customer or the NIP send policies to the network element in order to configure the active node's resources.

- Signalling: The policy target receives a reservation request, (an RSVP packet) and passes the request to the element manager.
- Self-adaptation: It is the PBANEM system itself that reconfigures the resources, after detecting a change on the network status.

2.5.1.2.2 Fault management

The fault management functionality needed is:

- Asynchronous events are sent from the active node to the PBANEM, in the case of an emergency.

  Precondition: The PBANEM should be able to register conditions to the active node through the RCF interface.
- Event correlation and event filtering mechanisms can be included, in order to reduce the management data and to optimise network performance dynamically.
- Policy conflict resolution.

2.5.1.2.3 Delegation management

It will be the case when an active node will not be able to accommodate all the resource allocation requests that may receive. In such a case, the management system will have to decide to accommodate part or all of the requested resources to a neighbouring node. What is required now, is an entity that is delegated the responsibility to enforce the policy to the other node. For example, an intelligent agent can be delegated the access rights from an active node, so that it can instruct another node to accept the incoming traffic.

Alternatively, the node that will be assigned the burden of accommodating the resource allocation request, can be of a different physical domain. In such a case the network level framework needs to be notified.

In any case, it is clear that the active node cannot possibly have any knowledge as to whether the neighbouring or any other nodes have sufficient resources. Probably, for nodes that are inside the same physical domain, the PDP manager can resolve this issue.

2.5.2 Specific issues for the architecture at network level

The generic architecture presented in the previous chapter has to be adapted to the specific functionality needed at each management level. In this section, we are going to study how this policy-based management architecture is adapted to Network Management Level functionality.

In chapter 2.3, a number of functionality requirements to the network management level have been stated. Most of them are intrinsically supported by the architecture proposed, i.e. dynamic extensibility of the functionality, delegation, customer specific policies support, etc. However, there are some requirements that are not intrinsically supported by the architecture proposed. These are, atomic set of policies support, network topology and resources monitoring, QoS route calculation, interdomain management, policy edition and network-to-element policy translation. The adoption of these requirements at the network level implies small additions to the functionality of the components of the architecture, and sometimes the addition of new components.

In the next sub-sections we describe in detail, how the functionality of the different components of the architecture have to be extended in order to support these network level specific requirements, and if necessary which new components have to be added to the architecture.

2.5.2.1 Policy Editor

The Policy Editor component has to be added to the architecture described in chapter 2.4, in order to support the edition of policies. This component offers a GUI both to the User Domain and to the Network Administrator. With this GUI the User (i.e. a NSP or another principal acting as a ANSP through delegation) will be able to edit policies to configure its already assigned resources (e.g. inject new service components to reconfigure the behaviour of the node against an application) and introduce them to the Network Management System with the appropriate format.

2.5.2.2 Monitoring System

The Monitoring System functionality specified for the architecture described, will fit basically the requirements needed at the network level, with the particularity that the resources to be monitored now, are network level resources (e.g. network topology, available network resources and status: links, routes, trunks...).

2.5.2.3 Resource Manager

The Resource Manager component supports a network level specific functionality (i.e. network topology and QoS route calculation) that is why we add it to the architecture presented in chapter 2.4.3. The Resource Manager is responsible of checking whether there are enough resources in the network as a whole for a new flow with QoS guarantees, and if so, which is the route for this flow. For being able to develop its functionality the Resource Manager will have to access the monitoring system, in order to obtain information about the network topology (physical and logical) and resource consumption. The network operator uses this component to clarify whether or not he has the appropriate resources for the execution of the service. In cases that it identifies that the consumer's request cannot be satisfied (lack of resources, low QoS etc.), he informs the User about the rejection of the service.

Network Resource might hold physical resource and logical resource. Network service requirements may first be considered as logical resource, then mapped to physical resource with using information of network topology and resource capacities.

2.5.2.4 Interdomain Manager

In a multidomain network, it is unrealistic and unscalable to assume a single point of administrative control for the entire network. A scalable alternative direction is to add a new component in the model that would be responsible for the interdomain interactions between the network operators of Fain Enterprise Model. This new component is the interdomain Manager. Interdomain Managers must be relied on bilateral communication for exchanging policy information between independent domains.

Each domain is placed under the administrative control of the Network Manager Node. Adjacent domains negotiate in order to determine the nature and extent of traffic that will traverse across their common boundaries. As part of this process, each domain describes its requested level of service to its neighbor's Network Manager Node through the interactions between the interdomain Managers. Interdomain Manager provides an admission decision based on its resource availability, bilateral financial arrangements, and the set of administrative policies in effect. In order to support the interdomain management interactions he must be in charge of requesting other domains resources (requester role) and answering to other domains requests (responder role) when interdomain services should be provided.
Consider, for example, a service level agreement (SLA) between two neighboring domains that specify a *premium service*, which guarantees packet delivery for a given traffic profile by reserving the required resources for this traffic. In order to support such a premium service, the Interdomain Managers in these domains should request form their PDPs to determine which PEP is going to enforce the *traffic control agreement* (TCA) to the appropriate edge PBANEM node, regarding resource reservation for the specific traffic that is represented by a set of traffic control parameters.

In the conceptual hierarchy of the PBNM system an SLA is a network-level element, whereas a TCA is a node-level element. In the interdomain context, edge PBANEM nodes that are at the boundary between two domains play a special role in the enforcement of the SLA between the two domains. The TCA is essentially the portion of the SLA parameters that relates to the role of a specific edge PBANEM as the network front-end.
The Interdomain Manager is conceptually a policy management entity located in a policy-enabled domain. It negotiates SLAs with neighboring domains and ensures compliance of the administered domain with the SLAs contracted. The Interdomain Manager performs three distinct tasks:

- Negotiation of SLAs with Interdomain Managers of neighboring domains
- Translation of SLAs. (The SLA Parser will be used for the translation)
- Delivery of the TCAs to the PDP.

Consider the following network topology where domain 1 represents an intranet, domain 2 a local SP, and domain 3 a large backbone (tier-1) SP. If we assume that the needs of domain 1 toward domain 3 are satisfied by a 64 kb/s flow of premium traffic, the following operations take place.
First, PDP1 (PDP of domain 1) requests a 64 kb/s SLA agreement. Next, the Interdomain Manager1 (Manager of domain1) requests the SLA from Interdomain Manager2. The Interdomain Manager2 contacts with the PDP2 to perform admission control based on whether there are sufficient premium resources available, and whether the commercial agreement allows such requests. The PDP will contact the Resource Manager and tell the Interdomain Manager the result. If the request is admitted, the Interdomain Manager2 requests from PDP2 to send(through PEP) a TCA derived from the SLA requested to the edge PBANEM node (its administered edge element node2) and responds positively to Interdomain Manager1, which forwards the decision to PDP1. This TCA models the traffic to be transferred from domain 1 via element node2. A similar TCA is sent by PDP1 to its administered edge element node1, instructing it to allow the given traffic to flow out to domain 2.
If premium resources are unavailable, or the request is not supported with the set of applicable commercial agreements and the domain's policies, PDP2 may decide to reject the
Request and notice PDP1 though the Interdomain Managers .The same applies to the relationship between PDP2 and PDP3. Such a process may be continued several domains further.
The edge node elements administered by the PDPs (such as element node1, element node2) perform several tasks in order to implement the kind of services (differentiated or integrated services).

2.5.2.5 PDP

The PDP presented in chapter 2.43 has to be slightly enhanced at the network level. Since the PDP is the core of the architecture presented, it needs to be enhanced to support the new components added to the architecture.

Thus, the PDP should be enhanced to be able to ask the resource manager for a route for a new flow with a certain QoS. This will allow the PDP to take a decision about whether a policy should be applied and in which nodes it should be applied.

The PDP has to be slightly extended to support interdomain functionality as well. It needs to interoperate with the Interdomain Manager component in order to be able to set Interdomain services for its customers, or other domains customers. Once the Interdomain manager informs PDP that an interaction with a different domain is going to take place PDP must assign the appropriate edge PBANEM node and determine the PEP, which is going to enforce the policy task for the implementation of the service.

Finally, the PDP functionality has to be enhanced in order to support atomic set of policies. The PDP has to recognise atomic PolicyGroups, and be able to treat them accordingly. That is, send the policies, when appropriate, to the correspondent PEPs in order to enforce them, and wait for the acknowledgement of the correct enforcement of all policies. If this acknowledgement is not received, the PDP is in charge of requesting to the PEPs the removal of all policies of the group.

2.5.2.6 PEP

The PEP functionality at the network level is, basically, the same as the one described in the architecture. That is, receive the policies to be enforced from the PDP and translate them into commands understandable by the policy targets. However, the policy targets of the network level PEP are the element level management systems. Therefore, the commands have to be sent in the form of the appropriate element level policies.

Summarising, the PEP at the network level is particularised to enforce the actions in the policy targets through the translation of the network level policies received into element level policies, and sending them to the appropriate element level management systems.

2.5.3 Interface of the Network Management Architecture to the Service Management Level

In the following, we will identify functional interfaces between Service Development/ Operations and Network/Systems Management System or processes in FAIN. The idea is to concentrate on the Fulfilllment and Assurance components and functions only. For a full description of the TMF TOM functions and processes the reader is refered to [2].

The figure below depicts the various processes of the TOM.

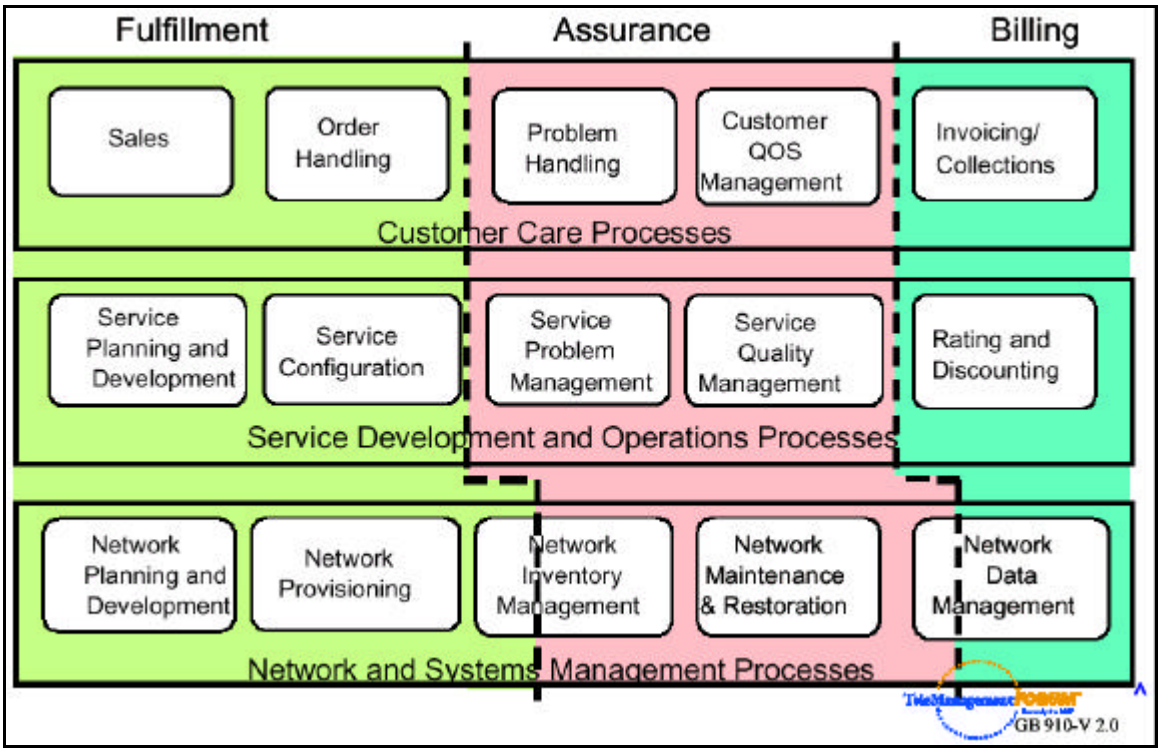

Figure 37 - Telecom Operations Map

The subprocesses of the Service level management are the following:

- *Service Planning and Development*: this process encompasses the planning and development of both unforecasted service requests and custom service requests. This process is also responsible for the planning and implementation of all that is necessary to ensure that the Service Development and Operations process performs effectively and efficiently: work center planning and realization, workforce management, training, etc.
- *Service Configuration:* this process encompasses the design, installation, configuration, end to end service testing and activation of service for customers; it also supports the re-configuration of service (either due to customer demand or problem resolution) after initial service installation.
- *Service Problem Management*: this process encompasses reporting on service problems and trouble performance, isolating the root cause of service-affecting and non-service-affecting failures and acting to resolve them.
- *Service Quality Management*: this process supports monitoring service or product quality and cost on a service class basis; it also encompasses taking appropriate action to keep service levels within agreed targets for each service class and to either keep ahead of demand or alert opportune processes to slow sales.
- *Rating and Discounting:* This process encompasses applying the correct rating rules to usage data on a customer-by-customer basis, as required for a usage based service, applying any discounts agreed to as part of the ordering process, applying promotional discounts and charges, applying outage credits, applying rebates or charges due because SLAs were not met or exceeded respectively, and resolving unidentified and zero billed usage cases.

For FAIN, Billing and Planning (for Fulfillment) are out of scope of the project. Therefor in Service Management we will consider only Service Development, Service Configuration, Service Problem Management and Service Quality Management.

Following interfaces to be provided by the Network and Systems Management (i.e. the Network Manager component) will be used:

- Open_management (out: sess_id)
- Set/get_configuration(in: sess_id, in: {monitoring, resource, pdp, pep}, out: configuration)
- Get_configuration_notification (in: sess_id, out: configuration)
- Distribute (in: sess_id, in: service_code, in: service_policies)
- Get_distribution_status (in: sess_id)
- Close_management (in: sess_id)

2.6 R12 - DESIGN OF TESTING ENVIRONMENT AND SCENARIOS

Active networks offer a means whereby new services and network management possibilities can be dynamically supported by networks. The advantages of such an approach has been discussed in detail in section 2.1. Having such dynamics is not without potential dangers however, e.g. the possibility of damaging or malicious code insertion, the potential for unforeseen side effects of the code insertion or the possibility of the inserted code not achieving the desired effects. Various approaches might be taken to help prevent such undesired behaviours, one that we investigate in this section is an approach based upon *testing*.

This chapter presents the design of a testing environment for PBANEM system. It is structured as follows: section 2.6.1 describes the objective of testing in order to demonstrate the advantages of AN-based management solution. Section 2.6.2 presents the FAIN methodology, a guideline for designing PBANEM testing system. Section 2.6.3 discusses an engineering scenario "Delegated QoS Management" and its test environment. 2.6.4 describes a Virtual Private Network scenario, proposing both network level management architecture and element level architecture.

2.6.1 Objective and Scope

From network management viewpoint, FAIN aims to deliver a flexible and customisable solution to ease management tasks. It enhances the IETF – Policy Based Networking (PBN) approach by Active Networking technology. Previous chapters present the PBANEM architecture, which is the core of the solution. Some benefits of such an active solution are explained in detail in previous chapters, e.g. *flexibility, extensibility, Automation of management tasks, Delegation, Application-specific management, Simplified service deployment, Benchmarking*, etc. A design of testing system aims to demonstrate these benefits as compared to conventional node management.

According to the workplan, in current phase we focus on experimenting active technology, e.g. active packets, to enhance existing policy-based network management approach. Management of Active Networks is rather proposed as future work when the infrastructure, e.g. the FAIN active nodes, is available. This restricts the scope of the current design as follows:

1. we focus on testing the implementation of the architecture and integration of its components into active node prototypes as available today, such as the ORB-based DPE platform or ABLE.
2. we separate management of heterogeneous prototypes, leaving interoperability test a future task.
3. benchmarking with existing PBN products or other AN-based PBN is a pending task.
4. we focus on testing the feasibility of the PBANEM architecture to realise such benefits as flexibility by active policies, application-specific management, delegation. Other benefits are left open, as they either require integration with another case study - ASP, or provision of a standard network resource interface as P1520-L [7]. These requirements are still unable to fulfil at this stage.

More specifically, we will define:

- the test infrastructure that consists of existing AN platforms. Software, hardware, and network configurations will be described.
- the test targets such as the management functions (e.g. configuration mgmt., performance mgmt., or fault mgmt.) and related policies (application-specific policy, QoS policy, VPN policy, security policy, etc.). These functions may be used in particular scenarios, e.g. QoS provisioning, to evaluate the strength and/or weakness of using active technology for network element management.
- the test cases including testing individual components, internal/external interfaces, the integrated system as specified in the PBANEM architecture. Among the interface tests, external interfaces will be the key focus. They include interface Int1, Int8, Int10.
- the added value tests, e.g. test the integration with PBNM, test the interworking and interoperation of active nodes.

Based on this plan we should be able to implement and exercise the test system straightforwardly. Scenarios will show how the PBANEM works in reality, and the benefits. However the ultimate goal of designing such a test system is to investigate the aspects that help us to evaluate/revise the initial PBANEM architecture, and the active node architecture as a whole. Some key aspects to evaluate are:

1. key functionality: translation of high-level SLA information into policy, and then into device configuration operations with no loss of semantics.

2. consistency of policy decision-making and enforcement across heterogeneous active nodes and execution environments.
3. interoperability among the deployed services to achieve router-independent PBANEM design
4. system/module extensibility by dynamic provision
5. system architecture that accommodates various policy provisioning means and a wide variety of policy, for example, application-specific policies and operator-managed QoS control policies
6. usefulness and restriction of using active packets to carry policy data
7. comprehensive requirements on the active node interface
8. security risk raised by allowing customer to manage a partial network via policy
9. conflict between policies to ensure the robustness of system
10. benchmarking measures, e.g. performance, scalability, etc.

2.6.2 CORBA-based Test Methodology

Testing in the active networking domain might be challenging due to the high complexity, which is introduced by risk-taking technical openness such as code injection, dynamic deployment, customer-involved management., etc.

This core test framework will initially be based around the testing of the CORBA based systems. This choice is taken due in part to the wide body of literature [18]-[22] in this area and also due to the expected technologies that are to be applied in FAIN. That is, it is expected that much of the functionality of the active nodes and their management subsystem will be defined using CORBA Interface Definition Language (IDL) specifications. Further the usage of these specifications, i.e. in implementations, will necessitate the adoption and usage of IDL. In addition, it is expected that the reference points being developed between the different roles/actors in FAIN by WP2 will also be specified in IDL together with appropriate usage/interaction diagrams.

The development of a core-testing framework for CORBA based systems, requires at a minimum that the testing of IDL interfaces can be achieved. IDL alone provides the basic syntax for communication between CORBA objects. It does not allow for[16] the specification of the ordering of operations, concrete values and pattern matching of parameters used to test interfaces, i.e. the most basic features required to test a system where the communication syntax is known. Various possibilities exist for including such extra information, Message Sequence Charts, UML being two. These two languages are developed primarily for reasons other than testing however, namely they were developed with emphasis on requirements analysis and software development more generally, and not explicitly with the intention of being languages for testing. The only full-standardised testing language is Tree and Tabular Combined Notation (TTCN) [23].

Testing can be regarded from numerous viewpoints depending upon how much information is available to the tester during the testing process. On the one hand, the tester might know only the IDL interface in which case, the system under test can then only be regarded as a black box. No knowledge of the internal functionality of the system under test is known. Alternatively, if the tester has some knowledge of the system under test in addition to the IDL description, e.g. knowledge of the allowed operation ordering or internal decomposition of the system under test then a more detailed and useful testing process can be achieved. This can be regarded as white or grey box testing depending upon the additional knowledge.

[16] Except for an informal support through comments in the IDL specification.

In terms of FAIN, the PBANEM under development can be regarded as white box testing. That is, the interfaces and behaviour of the components are completely available to the tester since in this case, the tester is the software developer. As a result, it should be the case that all components should be tested in FAIN. This includes components that are developed to run upon the active nodes, e.g. the components that the active nodes support, and components/code to be dynamically deployed on the nodes.

It should be noted that it is likely that the latter components will always require testing to be applied to them before being deployed. Whereas, the former components are more static and hence the testing process might be complete once the node implementation is complete, i.e. the deployment of code on the active node does not require that the whole node functionality is re-tested.

Put another way, testing an active node architecture might require ensuring that when a request on an existing management interface to reserve node resources arrives then the associated node resources can be successfully reserved[17]. Testing a component to be dynamically deployed who's functionality is to reserve node resources requires only that the appropriate node resource reservation management interface is correctly invoked with the correct parameters etc.

In work package 4, we focus upon the reservation and monitoring of node resources etc through a policy based approach. In addition, we extend the basic resource request scenario to multiple (application specific) resource request scenarios where the policies that are applicable to the users and applications are the deciding factors on the success of the resource monitoring or reservation requests, i.e. there might be contradictory or competing policies in place for network resources.

To conclude this summary, the testing scenario and associated testing environment in FAIN emphasise the testing of the active applications and not simply the testing of the active nodes themselves, although the testing of the active applications themselves necessitates a testing of a subset of the active node also to ensure the active applications demands are met or not.

Interworking test among various scenarios is proposed as future work. A detailed test plan will be proposed later to include testing internal/external interfaces, components, subsystems, integrated system and interoperation among heterogeneous implementations.

2.6.3 Engineering Scenario 1 – Delegated QoS Management

2.6.3.1 Test Environment

We plan to use existing active network platform to trial the initial PBANEM prototypes. [24] presents a state-of-the-art survey of such platforms. Among them, a DPE platform *OrbAN* is tentatively selected.

OrbAN as a DPE platform represents an Execution Environment (EE) to deploy and execute new network services or management functions. It is based on a modular ORB - Jonathan and runs on Java runtime environment. The interface it offers to new active application developer include those for resource access, code deployment, and QoS binding. In addition it implements an interface and GUI for management/operation purpose, e.g. monitoring the resource usage or bandwidth utilisation, deploying components, etc. The DPE platform runs on a proxy and controls the gigabit router GR2000. Therefore the proxy and the router together form a virtual active router. The proxy is normally a PC workstation running Linux, and also called router controller (rc).

[17] assuming of course that the request was a valid one, e.g. from a user allowed to reserve such resources

The platform is very flexible to implement a network/element management system. Firstly it provides a life cycle control of components, enabling dynamic plug-in or de-install of new management functions. With the policy-based interface for router control, it is quite straightforward to implement new functions like monitoring or resource allocation, etc. The natively supported policy is rule-based and can be mapped to XML-based policy format. As the interface provides rich features in configuring the QoS functions, e.g. packet classification, marking, priority-based scheduling, it supports advanced management functions other than mere MIB-based actions such as event trap or status polling. The platform has been used to develop scenarios for managing network resources, such advanced resource reservation, Diffserv-based bandwidth allocation, active IP metering, etc.

2.6.3.1.1 Software environment

- JDK1.2, Jonathan 2.0b5, Hypersonic SQL package (tested with hsql_143)
- OrbAN software including packages *ACM, QORB, Bootstrap, RouterResourceManager, GenericRouter, tools, Wrapper, ReA*, etc. For more information, refer to the API guide [25].
- GR2000 that offers a Command Line Interface for QoS configuration.

2.6.3.1.2 Network configuration

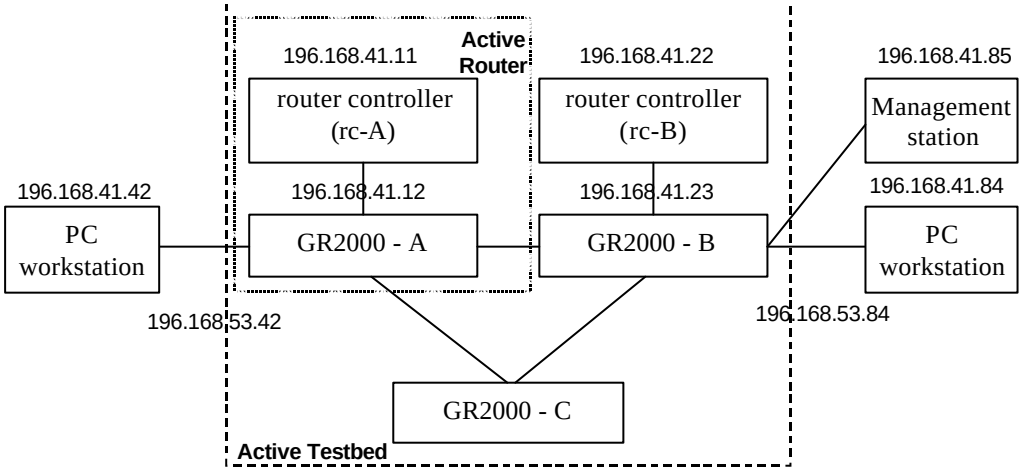

Figure 38 - Network Configuration for DPE Test Environment

2.6.3.2 Network QoS Provision

According to the methodology described previously, we focus on testing the application scenarios which implement the PBANEM architecture and demonstrates the tangible benefits. Therefore the generic architecture need to be specialised according to requirements of particular problem domain. Accordingly the component and interface specification are simplified and become more concrete to execute reasonable test procedure.

Integrated Services (Intserv) and Differentiated Services (Diffserv) are the well-established paradigms proposed by IETF to achieve end-to-end IP QoS for a wide variety of network applications. However Intserv is more appropriate for small private networks, but not suitable for use in the core of the Internet. In contrast, Diffserv was developed to scale well to large networks. They can be seen as complimentary [26]. The two approaches must be able to coexist and effectively inter-operate.

From the perspective of Intserv/RSVP, Diffserv regions of the network are treated as virtual links connecting Intserv/RSVP-capable routers or hosts. Within the Diffserv regions of the network, routers implement specific PHBs for aggregated traffic control. Intserv/RSVP network regions can be v referred to as 'customers' of the Diffserv network regions.

Using Intserv with Diffserv brings some key benefits such as:

- explicit admission control:. Diffserv networks can only provide 'implicit admission control', which might waste resource due to configuration of aggregated traffic forwarding. With interworking with Intserv, a network may accept or reject these requests in response according to resource availability for each flow. This is 'explicit admission control', which helps to assure that network resources are optimally used.
- traffic identification/classification: within Diffserv network regions, traffic is allotted service based on the Diffserv CodePoint (DSCP) marked in each packet's IP header. This requires classification by marking packet header fields. However, the classification criteria may change frequently. Reservation signalling such as RSVP or others is ideally suited for propagating the dynamic modifications.

A general interoperation framework is depicted in Figure 39.

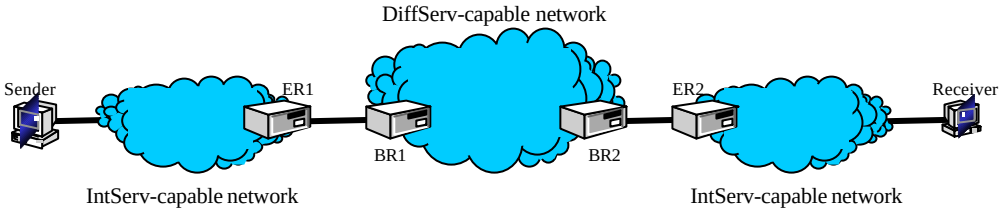

Figure 39 - Interoperatibility Reference Network Configuration

The reference network includes a Diffserv region interconnecting two Intserv/RSVP regions. In the interest of simplicity, a single QoS sender in one of the Intserv/RSVP network regions and a single QoS receiver in the other will be considered. Both sending and receiving hosts use Intserv/RSVP to communicate the quantitative QoS requirements of QoS-aware applications running on the host. It is assumed that RSVP signalling messages travel end-to-end between sender and receiver to support reservations in the Intserv network regions. It is required that these end-to-end RSVP messages are carried across the Diffserv region.

Intserv/RSVP signalling requests specify an Intserv service type and a set of quantitative parameters known as a 'flowspec'. Admission control agents for Diffserv network regions must map Intserv service types to a corresponding Diffserv service level (DSCP or PHB) that can reasonably extend the Intserv service type requested by the application. The admission control agent can then approve or reject resource requests based on the capacity available in the Diffserv network region at the mapped service level.

Routers at the ingress (entry) and egress (exit) points of the network are known as Edge Routers in Intserv and Border Routers in Diffserv. In Figure above, ER1 and ER2 are edge routers residing in the Intserv/RSVP network regions. BR1 and BR2 are border routers, residing in the Diffserv network region.

One big advantage with active networking is that functions can be dynamic provisioned. This implies that they can be dynamically installed, de-installed, configured or even programmed. The actors who initiate the provisioning procedure can be traditional roles like network operators and service providers, or more risky roles such as end-users. Moreover they have more control of the execution of these functions than today. A life cycle control of the code segment is possible, going far beyond static software management. This enables a more flexible control and management of networks, either resources or services. It also supports delegation of management overhead.

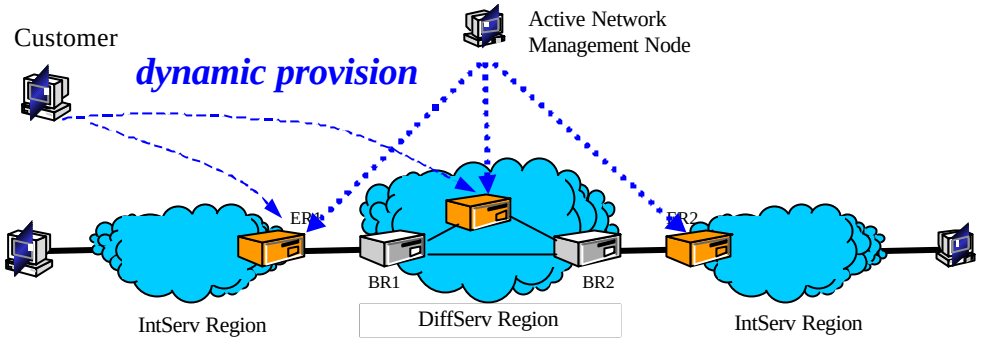

Figure 40 - Active Network for Intserv & Diffserv Interoperation

The Intserv/Diffserv interoperation framework may be enhanced with AN as depicted in Figure 40. In this case, the network equipment such as edge routers and core routers are enhanced with dynamic provisioning capability and can be controlled by end hosts or network management centre. They become *active network element.* This implies many benefits such as:

- new admission control algorithms can be installed, experimented, old algorithms can be updated.
- customised or advanced reservation protocols can be deployed where it is need
- PHBs can be updated, re-configured, or de-installed
- bandwidth brokers can be built as the management centre has the either static or dynamic knowledge of topology, services, resources, network load status, etc., to support network-wide admission control. Network resource usage/allocation may be optimised.
- application-specific requirements can be respected by re-configuration

Obviously it also implies many risks, for example:

- code from customers may be harmful for network security, e.g. it is maliciously representing denial-of-service-attack.
- resource access (computing resources and communication resources) need to be carefully controlled, as a number of functions share the access with potential conflict.
- code distribution mechanism need to be very flexible, and should not unnecessary restrict the size, language or capability of code.

For this reason, a strict control by the owner of infrastructure, e.g. network operators, should be enforced regarding the provisioning procedure, e.g. who is allowed to control which part of networks and how. In case other actors may control the network, a kind of SLA between network operator and them is required to be respected and enforced.

Policy is being widely used as a scalable and interoperable solution for managing network resources. QoS expectation of applications/end users can be achieved via carefully defined policies. The policies are enforced inside the network element and customise network state/mechanisms to deliver certain QoS. However, due to various requirements of QoS from a wide range of customers and their applications, it is unrealistic for a public network operator to define policies and enforce them in the network in a consistent and interference-free way. That justifies the observation that policy based network is currently popular for private networks. For large scale networks, delegation of QoS management is needed and described in next section.

2.6.3.3 Element-Level Management Architecture

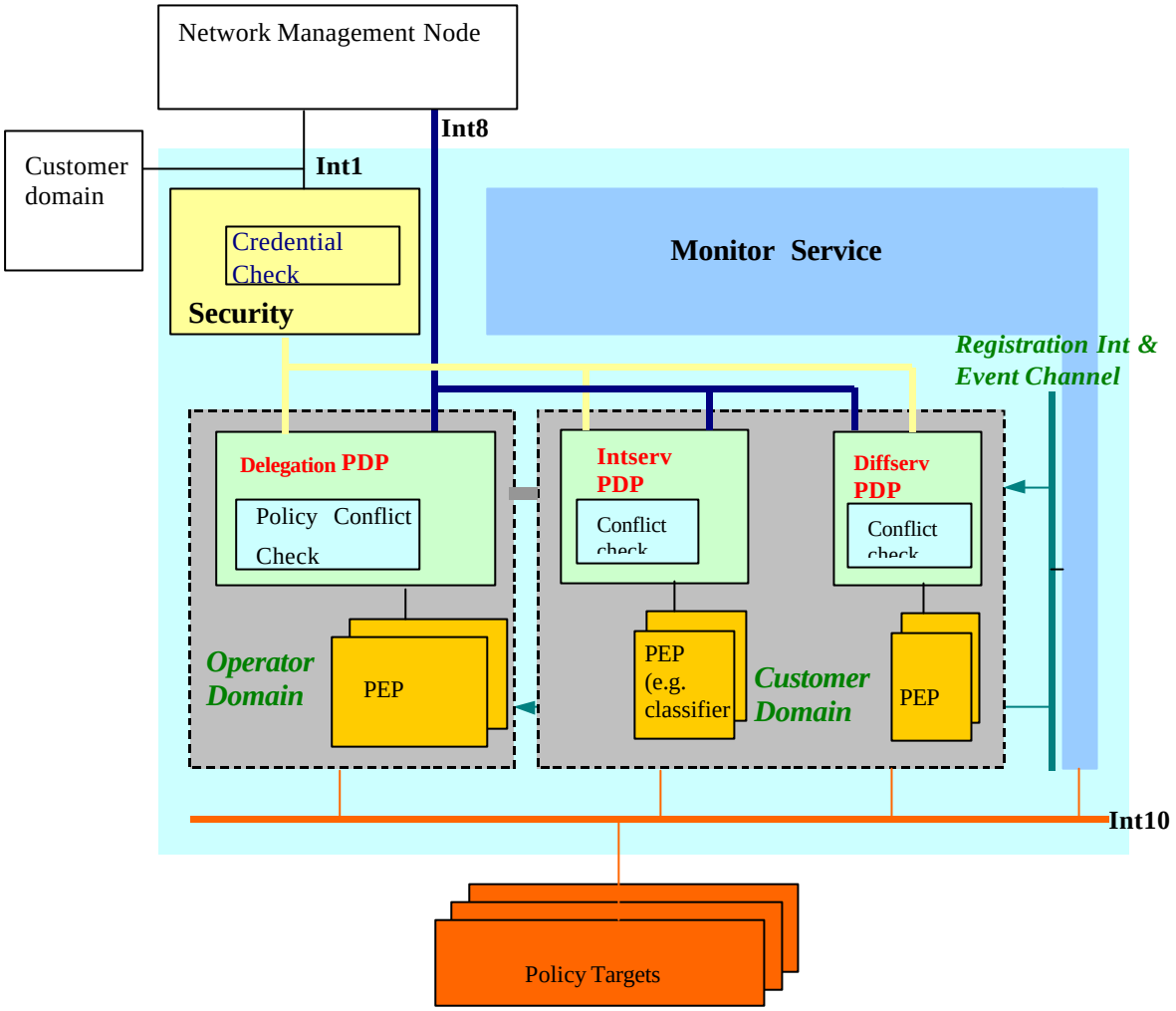

Figure 41 - Architecture for QoS Delegation Scenario

2.6.3.3.1 Delegation of Management Task

Management delegation is developed to ease management overhead of network administrators and centralised management system. A few technologies are used for this purpose such as automation by mobile agents or active packets. Routine tasks are thus moved and performed close to the device, e.g. a router. Redundant information transfer like SNMP status message is thus avoided. For the similar reasons, we need new technologies for delegating network resource management from network operators to customers, i.e. corporate users. One of the challenges we face to deploy quality of service on a large scale is to accommodate the diverse QoS requirements from different users, applications in a single end-to-end QoS framework. A feasible management model should enable network operators to be free of this overhead by, for example, securely delegating QoS provision process to those who require it.

A secure way of delegation should be based on virtual customer networks, a group of virtual network resources that is either statically or dynamically allocated to a customer. A customer is then authorised to allocate/use the resources to its users' applications in a customised fashion. A network operator would be responsible for provisioning these virtual resources and leaves managing of diverse QoS requirements to customers. More dynamically mechanisms can be developed to enable multiplexing of virtual resources by resource partitioning. Additionally, the network operator should ensure that in each network element, restricted access control is enforced to ensure different customers' QoS mechanisms do not interfere with each other. It is therefore a customers' task to provision QoS mechanism, define policies and enforce them for the applications. The risk failing to meet a QoS requirement will be limited to a customer's resource domain.

2.6.3.3.2 Functionality

Element management address managing the element resources, services, state, etc. The element management functions can be seen as active application components. They may be dynamic provisioned to perform FCAPS-like management tasks. Figure 41 depicts an example of such functions to deliver end-to-end QoS by interworking Intserv and Diffserv. This architecture represents an active edge router which sits on the boundary of Intserv and Diffserv. Therefore it contains resource reservation functions on the Intserv side, and per-hop behaviour on the Diffserv side. These comprises the PEP group which accesses the ANE to enforce the policy, e.g. set queuing priority on certain output port for a flow. A unified monitoring module contains generic functions such as timing service (timer) and metering block. Registration of conditions and event triggering are done through a registration interface and event channel. Both Intserv part and Diffserv part may have access to the monitor module. Policies are processed separately for Intserv and Diffserv domain. Potential conflicts and service mapping are resolved inside Intserv PDP and Diffserv PDP. The PDPs basically conduct admission control tasks.

A privileged delegation PDP controls the virtual network set-up, and configures the device to enforce access control. It is administrated by network operators via policies which translates a service level agreement of delegation into configuration commands.

Note that this architecture can be simplified as the active core router by keeping Diffserv functions, or a normal RSVP-capable router in the access network by keeping Intserv functions.

The internal interfaces among the security check block, PDP group, PEP group, and monitoring block need to be aligned with the design, at least partially. Additionally, we consider the local repository and access protocol an implementation decision, thus do not specify them here.

Meta-policy manager is implicit in the architecture. Its task is mainly checking the policy requests against existing meta-policies, which specify who is allowed to deploy what type of policies. It is a key function in the security block.

The resources inside an active network element are policy targets. They may have internal policy-enforcement mechanisms, in which case a mapping from XML-based policy representation to device-specific policy representations is necessary. Control the targets is through interface Int-10 as specified previously.

2.6.3.3.3 QoS Policy Information Model

A policy domain is designed to support the QoS Mgmt. Delegation scenario using the multi-PDP PBANEM architecture. It is derived from the core FAIN information model by adding class definitions which is scenario-specific. As a policy definition specifies the actions to take for particular functional elements when certain conditions are met. Therefore it is prerequisite to have a standard definition of functional elements, and parameters applicable to them. For this the "RSVP Policy Control Criteria PIB" [72] and "Diffserv PIB" [73] are our reference specification.

The scenario makes use of active technology and tries to manage the resource allocation on a per-customer basis. Overhead of the management is delegated to individual customer to fulfil individual users' bandwidth need. Policy system modules (e.g. a QoS PDP) can be dynamically provisioned via interface with the service provisioning framework. It aims to demonstrate the advantages of active networking to ease network bandwidth management procedure.

In this scenario, a policy domain is supposed to support the following functions:

- QoS provisioning: Intserv and Diffserv will be supported, therefore the QoS policy information model [83] is considered a standard model and will be reused and extended, if necessary.
- Delegation: a network is partitioned into virtual networks, management of virtual resources are delegated to customers according to SLAs. Policies will be defined and transferred from network managers to network elements, and perform partitioning and set-up delegation parameters, e.g. customer credential for authentication.
- Provision: the policy system modules will be dynamically downloaded and updated, if possible. Policies need to specify how this can be done for those QoS modules such as an Diffserv PDP, its traffic conditioning blocks, metering block, etc.

In the following, classes are defined according to these function categories.

2.6.3.3.3.1 QoS Policy

The class hierarchy of QoS policies within FAIN policy core information model is depicted as below. The proposed extensions are highlighted.

```
 [ManagedElement] (abstract)
  |
  +--Policy (abstract)
  | |
  | +---PolicyAction (abstract)
  | |    |
  | |    +---fainSimplePolicyAction
  | |    |  |
  | |    |  +---fainQoSPolicyRSVPSimpleAction
  | |    |
  | |    +---fainQoSPolicyDiscardAction
  | |    |
  | |    +---fainQoSPolicyAdmissionAction
  | |    |  |
  | |    |  +---fainQoSPolicyPoliceAction
  | |    |  |
  | |    |  +---fainQoSPolicyShapeActionAction
  | |    |  |
  | |    |  +---fainQoSPolicyRSVPAdmissionAction
  | |    |
  | |    +---fainQosPolicyPHBAction (abstract)
  | |       |
  | |       +---fainQoSPolicyBandwidthAction
  | |       |
  | |       +---fainQoSPolicyCongestionControlAction
  | |
  | +---fainQosPolicyTrfcProf (abstract)
  | |  |
  | |  +---fainQosPolicyTokenBucketTrfcProf
  | |  |
  | |  +---fainQosPolicyIntServTrfcProf
```

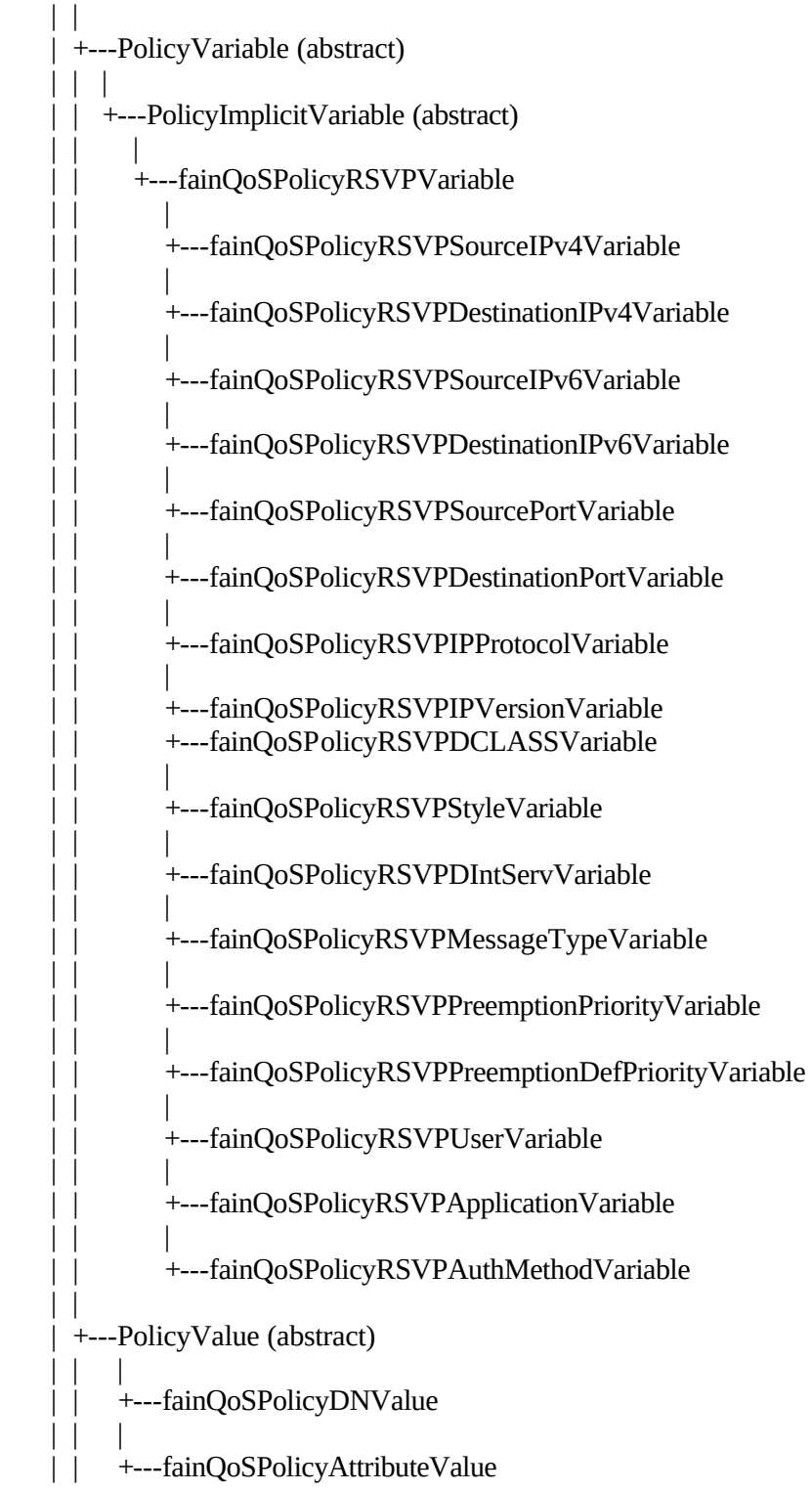

Figure 42 - QoS Policy Class Hierarchy

Most of these classes is same with QoS Policy Information Model (QPIM). For a more detailed description of each class and its properties, refers to [38].

2.6.3.3.3.2 Delegation Policy

2.6.3.3.3.2.1 fainNetPartitionAction

NAME fainNetPartitionAction

DESCRIPTION This class specifies the partitioned network configuration to be allocated to a customer

DERIVED FROM fainSimplePolicyAction

ABSTRACT FALSE

PROPERTIES topologyVirtualNet, bandwidthVirtualNet

2.6.3.3.3.2.2 fainElementPartitionAction

NAME fainElementPartitionAction

DESCRIPTION This class specifies the partitioned element configuration to be allocated to a customer

DERIVED FROM fainSimplePolicyAction

ABSTRACT FALSE

PROPERTIES idVirtualNet, idInterface, idOutputQuque, bandwidthPercentage, priority

2.6.3.3.3.2.3 fainDelegationAction

NAME fainDelegationAction

DESCRIPTION This class specifies the delegation parameters to be configured for a customer

DERIVED FROM fainSimplePolicyAction

ABSTRACT FALSE

PROPERTIES customerCredential, accessControlEnabled

2.6.3.3.3.2.4 fainCustomerIDVariable

NAME fainCustomerIDVariable

DESCRIPTION The identifier of a customer.

ALLOWED VALUE TYPES: PolicyStringValue

DERIVED FROM PolicyImplicitVariable

ABSTRACT FALSE

PROPERTIES credential, certificate

2.6.3.3.3.3 Provision Policy

2.6.3.3.3.3.1 fainQoSPDPProvisionAction

NAME fainQoSPDPProvisionAction

DESCRIPTION This class specifies the additional parameters for deploying, updating and reconfiguring QoS policy decision modules realizing QoS management

DERIVED FROM fainServiceCodeUpdateAction

ABSTRACT FALSE

PROPERTIES customerCredential, idPEPgroup, idMonitor

2.6.3.3.3.3.2 fainQoSPEPProvisionAction

NAME fainQoSPEPProvisionAction

DESCRIPTION This class specifies the additional parameters for deploying, updating and reconfiguring QoS policy enforcement modules realizing QoS configuration

DERIVED FROM fainServiceCodeUpdateAction

ABSTRACT FALSE

PROPERTIES customerCredential, idPDP, idMonitor

Here we will give some examples of policy and explain how it is used to provide network-level QoS. They are tentatively specified as below, and will be represented using XML.

2.6.3.3.4 External Interface Int-10

In addition to the interface specified in chapter 24, we propose the following resource abstractions which represents more service-specific abstractions [70] to support Intserv and Diffserv:

QoS control function		
TransmissionController	Determines the queue mode, thereby controlling package precedence on the interface output queue. Field: queue_mode (PRIORITY, ROUND-ROBIN, BANDWIDTH)	Component
QueueController	When there is a backlog on the output queue, packets remaining on the queue are transmitted or discarded depending on priority. Field: queue_size, enable_priority	Component
FlowController	Identify input IP packets by their flow-control settings, determine priority, rewrites TOS, and manages reserved bandwidth, among other operations.	Component
IP routing		
Filter	The function relays or discards packets that match a specific condition and relays all packets that do not match the condition. This feature helps maintain the security system.	Component

Table 2 - Resource Abstractions Table

2.6.4 Engineering scenario 2 – VPN

2.6.4.1 Overview

In this section we describe the network level of the VPN scenario developed within the FAIN project. The VPN Scenario uses Policy Based Management technology in order to establish, monitor and cancel a VPN for a customer using mobile agents and active networks technology. The management system developed in the FAIN project and demonstrated with this scenario is decomposed in two network levels, the network management level, and the element management level.

In this scenario, the main functionality of the network level VPN PDP and PEPs introduced within the PBANM, is that of receiving, through the offered GUI the policies from the customers, check whether this policies can be applied, and when they should be applied, translate this network level policies into element level policies understandable by the element level, and finally send these policies to the appropriate PBANEM nodes in the form of an XML file.

The functionality and components of the element level PDP and PEPs developed for the VPN scenario will be described in the next section.

2.6.4.1.1 Mobile Agent

The mobile agent paradigm intends to bring an increased performance and flexibility to distributed systems by promoting "autonomous code migration" (mobile code moving between places) instead of traditional RPC (remote procedure call). With code migration, the actual code or script moves from place to place and executes locally, achieving lower latency, little need for remote interactions and highly flexible control.

Most of the communicative mechanisms such as CORBA may be able to actually change the behavior of other agents by sending messages. But mobile agents move their data and code (especially the code) to a remote site and execute locally. Mobile agents can also be used to modify or introduce new behavior by executing the control carried by code in mobile agent.

Mobile agents are widely used in telecom and network management. They are very effective as they can take over the burden of the complex interaction mechanisms between different network players, such as negotiations or new service injection. Mobile agents can easily represent one of the business roles such as backbone operator, access provider, service provider or end-user, and act on their behalf, based on established policies.

Mobile agent technology is also used in FAIN project. Examples of mobile agent platforms are Odyssey (General Magic), D'agent/Agent TCL (Dartmouth College), Voyager (ObjectSpace), Aglets (IBM) and Grasshopper (IKV++), of which Grasshopper will be used in FAIN.

Grasshopper [42] is a mobile agent development and runtime platform which is built on top of a distributed processing environment. This achieves an integration of the traditional client/server paradigm and mobile agent technology. Grasshopper is implemented in Java, based on the Java 2 specification. Most importantly, Grasshopper has been designed in conformance with the first mobile agent industry standard given by OMG, namely the Object Management Group's Mobile Agent System Interoperability Facility (MASIF) [43], which allows interoperability of different mobile agent platforms and the deployment of mobile agents on CORBA environments. In addition, the latest Grasshopper version is also compliant with the specifications of the Foundation for Intelligent Physical Agents (FIPA) [44]. Grasshopper is also the agent platform of choice in multiple international research projects within the European CLIMATE (Cluster for Intelligent Mobile Agents for Telecommunication Environments).

Here in this architecture and scenario, Grasshopper based mobile agent technology is used to transport policies from PDP to PEP and also initiate the enforcement of policies in PEP. Policy transport from network level to element level can also been implemented by mobile agent. But XML-RPC is used here for the XML-based policy transport between network level to element level, according to the suggestion from UPC who is responsible to the network level management.

2.6.4.1.2 XML syntax for Policies

The policies between the network level and the element level are expressed in XML [36]. The XML document will contain the necessary classes describing the policy rule This XML document will be carried by a Mobile Agent, which will carry other important information, basically security related information. However, in a first implementation stage of this scenario, the policies will be moved from the network level to the element level using an XML-RPC approach for simplicity reasons.

XML-RPC is a Remote Procedure Calling protocol that works over the Internet, using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned. An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML.

Using XML as language for expressing policies has several advantages [34]. XML is ideal for transferring information between heterogeneous platforms because XML parsers are available for many platforms. Another advantage is that XML policy documents can be validated against an XML policy schema that resides on a remote, trusted server because XML document can carry a reference to their XML Schema, instead of the Schema itself. Moreover, the policy syntax checking functionality, is done intrinsically by the XML parser through the validation of the XML policy against its XML schema.

2.6.4.2 Network-Level Management Architecture

The architecture of the PBANM with the VPN PDP and PEPs developed for the VPN scenario at the network level is given in the figure below. It conforms with the PBANM architecture described in chapter 2.4, with the network level specific functionalities discussed at chapter 2.5. The components are discussed in subsequent sections.

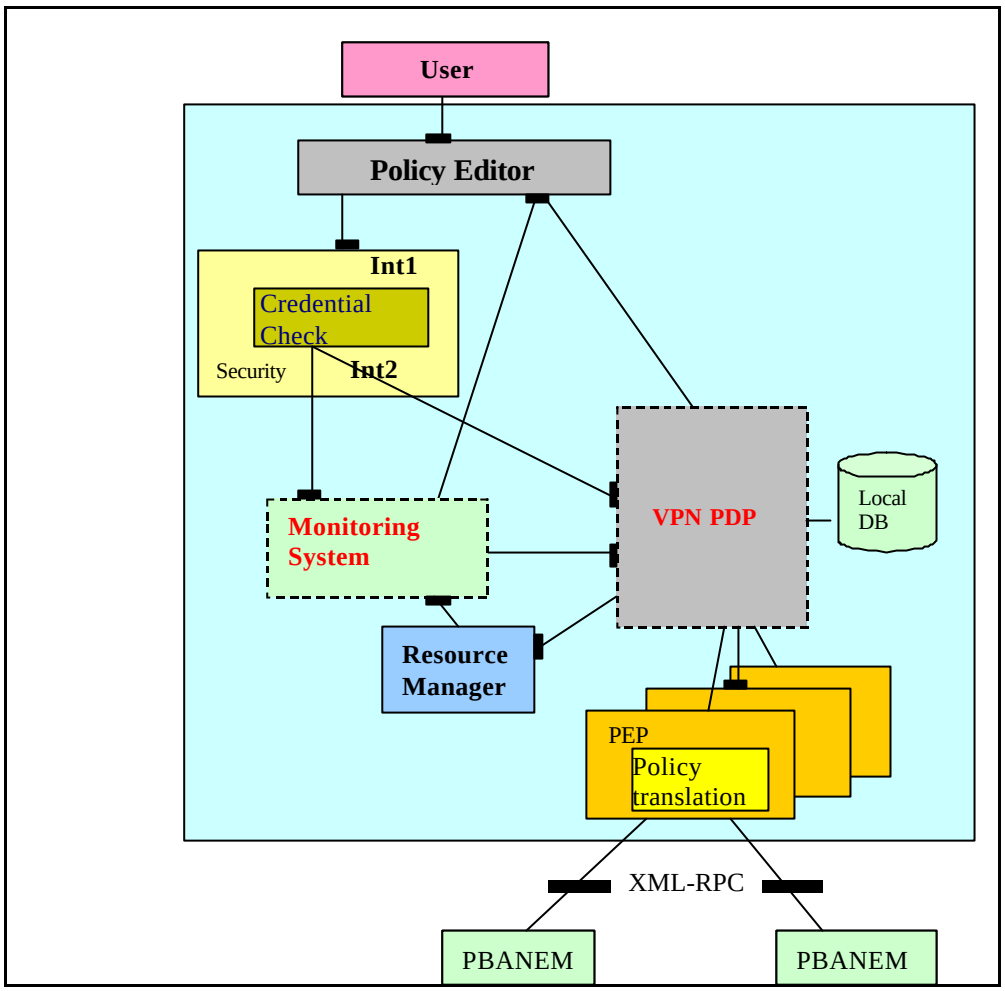

Figure 43 - VPN PBANM Scenario Architecture

2.6.4.2.1 Functionality and Components

2.6.4.2.1.1 Policy Editor

This component offers a GUI both to the Use. With this GUI the User will be able to edit policies to establish and cancel a VPN, and introduce them to the Network Management System with the appropriate format. The GUI will be guided to ease the introduction of the necessary data.

2.6.4.2.1.2 Credential Check Component

The Credential Check Component receives the policy and a user's credential from the policy editor. It is the responsible of checking the privileges of the user for requesting the creation of a VPN with a certain QoS. In order to develop this task, the credential check component takes this credential and looks in the metapolicy database for a metapolicy related with that credential. The Credential Check component retrieves the policy and checks if the intended management actions (policies) are available to the actor that presented the credential. If the checking is successful the policy is passed to the PDP. In a first stage of the implementation the policy conflict check component is not included.

2.6.4.2.1.3 Monitoring System

The Monitoring System will be responsible of monitoring network level resources (e.g. network topology, available network resources and status: links, routes, trunks...). As described in chapter 2.4 the PDPs will be able to register in the monitoring system the events they are interested in. The Resource Manager will need to access the monitoring system as well to recompile information from the network resources.

2.6.4.2.1.4 Resource Manager

The Resource Manager component supports a network level specific functionality (i.e. network topology and QoS route calculation). The Resource Manager is responsible of checking whether there are enough resources in the network as a whole for a new flow with QoS guarantees, and if so, which is the route for this flow. For being able to develop its functionality the Resource Manager will have to access the monitoring system, in order to obtain information about the network topology(physical and logical) and resource consumption.

The Resource Manager will receive a request from the VPN PDP to find the needed resources and the route for creating the VPN.

The implementation of the whole Resource Manager functionality will be progressive through the different implementation scenarios for simplicity reasons.

2.6.4.2.1.5 PDP

The VPN PDP will receive the policy from the Credential Check component. Then, it has to decide when this policy should be applied. In order to realise that, the PDP will look the conditions of the policy and decide whether it needs any information, in the form of events from the monitoring system, in order to make a decision. We pretend to use ILOG Rule software as decision module within the PDP [41], at least in an advanced stage of the implementation. If so, it registers the conditions in the monitoring system. Otherwise, e.g. the condition is a filter condition, it asks to the resource manager if there are enough resources to apply this policy.

If the answer from the Resource Manager is positive, the PDP stores the policy in the database and pass it to the appropriate PEP the policy along with the PBANEM nodes where it should be applied (these nodes are obtained from the Resource Manager response).

2.6.4.2.1.6 PEP

The PEP functionality at the network level is, basically, the same as the one described in the architecture. That is, receive the policies to be enforced from the PDP and translate them into commands understandable by the policy targets. However, the policy targets of the network level PEP are the element level management systems. Therefore, the commands have to be sent in the form of the appropriate element level policies.

The PEP will receive the policy from the PDP along with the PBANEM nodes where it should be set. Then, the PEP will translate the network level policies into the corresponding element level policies in XML. Moreover, it will send the element level policies to the correspondent PBANEM nodes using XML-RPC in a first stage, and mobile agents afterwards.

2.6.4.2.1.7 Database

The meta-policy database and the local policy database will be logically different, because they store semantically different policies, but they will be a unique physical database that can be accessed by the components of the PBANM system. Meta-policies are policies that will control the access to functionality and the behaviour of the PBANM system, while policies control the access and behaviour of the network resources managed by the network management system.

2.6.4.2.2 Working Flow

The steps shown below describe the interaction between different components in the scenario:

- Step 1: The User edits the VPN network level policies using the offered guided GUI.
- Step 2: Once the policy edition is finished, the Policy Editor component passes the network level policy to the Credential Check component along with a credential of the user who introduced the policy.
- Step 3: The Credential Check component checks whether that user is allowed to request those VPN resources, using the metapolicy database. If the User is not allowed an error message is sent back to the Policy Editor component that will reflect it in the GUI; otherwise the policies are passed to the VPN PDP component.
- Step 4: The VPN PDP component will look if any information from the monitoring system is needed in order to decide when the policy should be applied. If any information is needed, the PDP registers the event in the monitoring system and stores the policy in the database. Otherwise, the PDP requests to the Resource Manager whether there are enough free resources to set that policy.
- Step 5: The Resource Manager will receive the request from the PDP and look for an available route with the requested resources. If successful, the Resource Manager returns a positive response along with the route found. Otherwise, it returns a negative response.
- Step 6: The PDP sends the VPN policy along with the calculated route to the appropriate PEP.
- Step 7: The PEP translates the network level policy into element level policies and sends them to the corresponding PBANEM nodes, according to the calculated route. The policies are expressed in XML and sent using XML-RPC for simplicity reasons. In more advanced stages of the implementation the XML policies will be transported using mobile agents.

2.6.4.3 Network Element Management Architecture

The architecture of mobile agent based PBNEM is depicted as Figure 44, which fully conforms to the PBNEM structure in FAIN given in chapter 2.4.3. The components are discussed as follow.

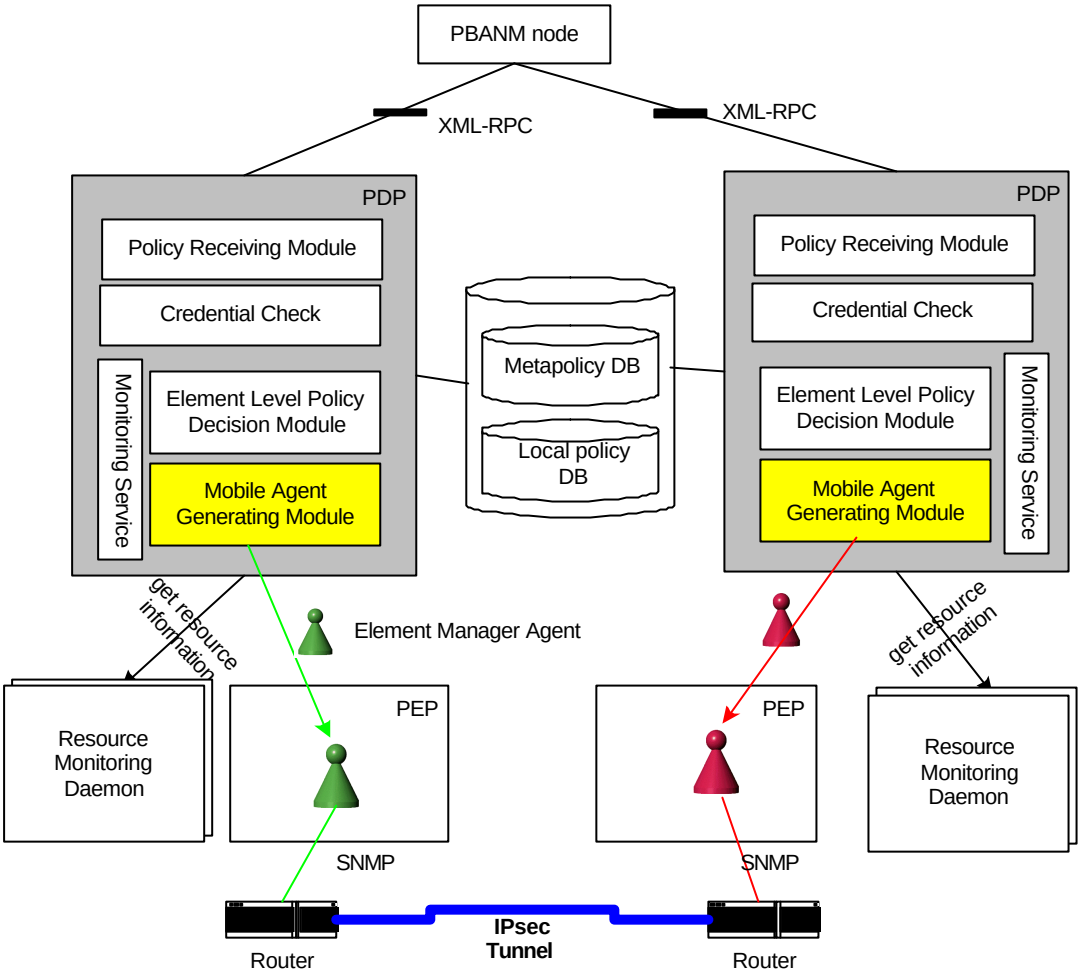

Figure 44 - Mobile Agent Based PBNEM Architecture

2.6.4.3.1 Functionality and Components

2.6.4.3.1.1 Policy Receiving Module

This module is implemented as a daemon for receiving XML-based policies given by network level. XML-RPC [40] protocol is used for policy transferring between two levels.

XML-RPC is a Remote Procedure Calling protocol that works over the Internet, using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned. An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML.

2.6.4.3.1.2 Credential Check

The credential check component will be in charge of checking the privileges for VPN establishment and cancellation granted to any actor.

Each actor that wants to setup or shut down VPN should also submit a credential. The Credential Check Component then takes this credential and looks in the meta-policy database for a meta-policy related with that credential. The Credential Check component retrieves the policy and checks if the intended management actions (policies) are available to the actor that presented the credential. Finally, if the credential is correct and the actor has the correspondent privileges these policies are passed to Element Level Policy Decision Module. Policy Conflict Check component is omitted in this scenario for simplicity.

2.6.4.3.1.3 Monitoring Service

The monitoring service component receives the registration of resource monitoring according to the requirement of policies and make sure that all the resources registered can be monitored. If the necessary metering block (daemon) is not currently instantiated it will try to download it by making a query to the Active Service Provisioning service and install it. But here in this scenario, all necessary resources monitoring are initiated previously due to the lack of RCF from WP3. Monitoring Service component just enquiries the Resource Monitoring Daemon to get the necessary information.

2.6.4.3.1.4 Resource Monitoring Daemon

The Resource Monitoring Daemons always exist on active nodes. It can be provided by RFC from WP3. In this scenario, these daemons are implemented by SNMP polling.

2.6.4.3.1.5 Element Level Policy Decision Module

This component can translate network level policy into element level policies, with the information from monitoring service.

After receiving the policies in XML file which also has passed the credential check, the Element Level Policy Decision Module extracts from the XML file the element level policies; then, it has to decide when this policy should be applied. In order to realise that, the module will look the conditions of the policy and decide whether it needs any information, in the form of events from the monitoring system, in order to make a decision. If so, it will ask Monitoring Service to register the conditions in the Monitoring Daemon. Otherwise, it asks the resource Monitoring Service if there are enough resources to apply this policy. If the answer is positive, the policy will be passed to Mobile Agent Generating Module to be fulfilled.

All the policies are based on fixed schema so that they are understandable by different levels.

2.6.4.3.1.6 Mobile Agent Generating Module

Based on the parameters given by Element Level Policy Decision Module, this module can generate corresponding mobile agents to transport the policy to relative PEP and fulfil the policy.

2.6.4.3.1.7 Mobile Agent and its Platform

Based on the element level policies generated above, mobile agents bearing the PEPs that are in charge of corresponding policy as destinations are created automatically. These mobile agents migrate themselves to the specific PEPs to enforce the policy.

After arriving at the PEP, mobile agent, also served as SNMP wrapper at this moment, uses SNMP to set up the IPsec VPN.

After getting the return code of SNMP command execution, mobile agent goes back to PDP to inform the result, then removes itself automatically.

Here all mobile agent execution environment (i.e. Grasshopper Agency) needed for mobile agent life cycle is pre-installed on every PEP. If there is not mobile agent EE on the destination EE, active node mechanism such as ABLE++ can be used to transfer and setup the MA EE. Agency resides in a machine next to that element.

A Region Server, which provides the agents with the necessary information and location of the respective agencies, is also needed.

In addition, wrappers for the SNMP protocol had to be developed to enable the agents to communicate to the managed elements via SNMP. For simplicity, the functionality of wrapper is also implemented by mobile agent in this scenario, in which Advent SNMP Driver is used.

2.6.4.3.1.8 Database Components

The meta-policy database and the local policy database will be logically different, because they store semantically different policies, but they will be a unique physical database that can be accessed by the components of the PBANM system. Meta-policies are policies that will control the access to functionality and the behaviour of the PBANM system, while policies control the access and behaviour of the network resources managed by the network management system.

Plain files are used for policy database in this scenario.

2.6.4.3.2 Working flow

The structure depicted in Figure 44 shows the normal behavior of our policy based management system for setting up VPN, where the network manager node sends a set of provisioning policies to be set in the active network node.

The working flow and interaction between components in this scenario will be:

Step 1: The network manager sends a set of policies to the network element manager (actual PDP in element level) along with a credential.

Step 2: The Policy Receiving component receives the policies and checks if the actor with that credential is allowed to set policies, using the meta-policies database. If the credential is incorrect then an error message is sent back to network level; otherwise the policies are passed to the Element Level Policy Decision Module.

Step 3: After receiving the policies in XML file, the Element Level Policy Decision Module extracts from the XML file the information for element level, therefore getting the element level policies, which are transferred to related managed elements. All the policies are based on fixed schema so that they can be understood by different levels.

Step 4: According to the policies generated in step3, mobile agents bearing the PEPs that are in charge of corresponding policy as destinations are created automatically.

Step 5: These mobile agents migrate themselves to the specific PEPs. Here all mobile agent execution environment (i.e. Grasshopper Agency) is pre-installed on every PEP. If there is not mobile agent EE on the destination EE, active node mechanism such as ABLE++ can be used to transfer and setup the MA EE.

Step 6: After arriving at the PEP, mobile agent, also served as SNMP wrapper at this moment, uses SNMP to set up the IPsec VPN.

Step 7: After getting the return code of SNMP command execution, mobile agent goes back to PDP to inform the result, and removes itself automatically.

2.6.4.3.3 Testing bed

A test bed consisting of several IP routers, switches and machines was set up, which depicts the large scale network scenarios in some extents.

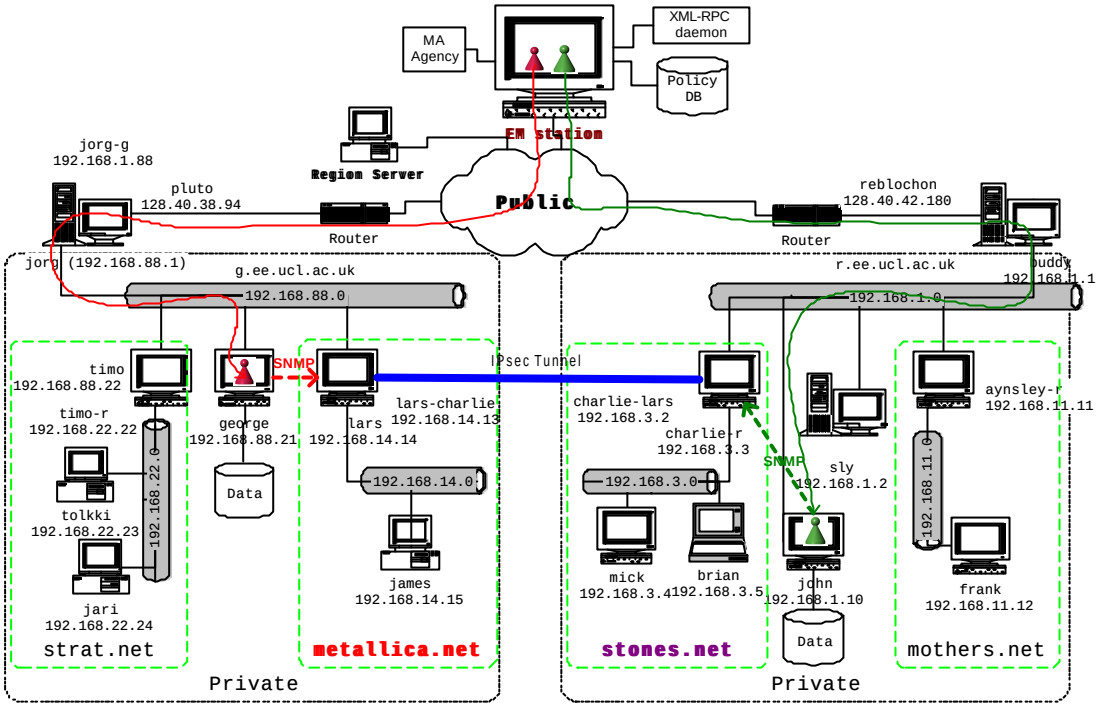

Figure 45 - Test bed for PBVPN

Both EM station and Region server reside on public network. EM station can receive XML-based policies from network level via XML-RPC daemon (here we use one EM station to receive two probably different element level policies for the sake of simplicity), then it creates two mobile agents which move themselves to george in g.ee.ucl.ac.uk private network and john in r.ee.ucl.ac.uk private network respectively. The mobile agent on george communicates with lars via SNMP to setup the right end of VPN, whilst the mobile agent on john sets up the left end of VPN on Charlie following the same procedure and method. Therefore, the IPsec based VPN is set up between lars and Charlie.

The cancellation of VPN follows the same procedure as the establishment of VPN.

2.6.5 Conclusion

This section presents two engineering scenarios which specialises the generic policy system architecture. We propose to implement these scenarios and use them to evaluate and justify the design of active node design and management system architecture.

2.7 R12 - CONCLUSIONS

Through this chapter, the initial Management Architecture designed in the scope of the FAIN project has been presented, both at the element and the network level. The approach taken is that of a two-level Policy based management system. The architecture has been designed in order to face Active Networks specific issues, and furthermore to take advantage from them.

The initial management architecture defined in this document ma be refined as the FAIN project evolves. Several issues may be further discussed and subject of discussion and, maybe, refinement. Some of them are summarised below.

- *FAIN Roadmap*: As a first step we should try and enhance the already existing routing solutions (GR2000, Cisco) and then try and provide active management solutions to active routers. Thus, we should include the PEPs at the element management level. In the second phase however, we may need to include the PEPs in the active routers. To make them even smarter, a local PDP module can also be part of the active router in the future.
- *Multiple PDPs granularity:* The need for multiple PDPs can be seen as the need to accommodate a rather complicated scenario that requires the combination of policies from different technical domains. For example an end-to-end QoS scenario needs both Intserv and Diffserv PDPs, as suggested by GMD. This is the case where one PDP exists per technical domain, but because there are more than one technical domains per management system, overall, we have multiple PDPs. Another possibility is the need for multiple PDPs per technical domain. For instance inside the Diffserv technical domain, another PDP may handle the access control (i.e enforce security policies). But neither of these cases is mandatory. There may be situations where the use of only one PDP per management system can be sufficient. What is needed however, is the ability to install new PDPs upon request.
- *PDP manager:* It is quite open at this stage to discuss the functionality of PDP manager. Obviously its role is to co-ordinate the PDP to support integrated scenarios, and resolve possible conflict. But it appears very scenario-specific, and there is no need to address this issue in FAIN, whose goal is to demonstrate how policy can be used to flexible manage the complexity of active network elements. With respect to policy conflict, PDPs for different scenarios/technologies manage different set of physical/virtual resources. A resource control framework on the node level will provide required access isolation, thus eliminate the conflicting probabilities. As a conclusion, this is a research issue for next phase of FAIN. As said above the PDP manager will:
    - Resolve inter-technical domain conflicts.
    - Assist in the management by delegation.
- *PDP Interoperation:* Again, inter-PDP communication is needed, but may be very technical domain-specific. The interaction very likely involves translation of policy behavior between different domain, e.g. maximum bandwidth guarantee in Intserv domain and code point in Diffserv domain. We won't expect a policy to be transferred through the communication. As another domain won't understand a policy from another domain anyway. Therefore Policy Behavior Translation will be the key part of the communication, and need to be managed both on element and network–wide level.

    1) Interoperation between PDPs inside the same technical domain.

    Helps in resolving conflicts that exist in the same technical domain.

    2) Interoperation between PDPs of different technical domains

    Done through the PDP manager. Helps in resolving conflicts that exist between different technical domains.
- *Dynamic Provisioning of PDPs*: We believe it is a straightforward design decision to let ASP care about the deployment process, be the component a PDP or PEP, or whatever. They naturally are active components. ASP will provide an interface to activate the provisioning. It will implement a node-local interface to PBANEM for activating code downloading. But PBANEM only does this routine task on request from PBNEM, without local decision.

- *Dynamic Provisioning of PEPs:* Extensibility is provided by the ASP feature. As long as we provide a modular architecture. The PBANEM design does not impact the extensibility of the system. The complexity of dynamically deploying composition of module is outside FAIN scope. The initial feeling is that PDP will be close coupled with PEPs to support policy behavior, and need to be deployed together. Address individual extensibility is missing the point.
- *Conflict resolution:* If there were no multiple PDPs inside the single technical domain or inside the PBANEM in general, the situation would be straightforward. There would be no chance of policy conflicts-at least at the element level. But in the case of multiple PDPs, this danger arises. This is because the different PDPs can send controversial policies to the same policy targets (physical or logical resources). The conflict possibility inside the technical domains, can be accommodated by communication between the PDPs. The inter-technical domain policy conflicts can be accommodated by an entity such as the PDP manager. For conflicts that involve physical domains, the need for the PBANM (network level management) to resolve them is necessary.
- *Event granularity:* PDP's can subscribe to events, ok, but ¿who sends the monitoring policies to be enforced?. Probably, the user will. However, the PDP has to subscribe to the event of its interest, we have to discuss the level of granularity at which the PDP can choose an event-> as higher less event types but the PDPs will receive more uninterested events, which will make them consume resources. We have to think carefully on how the monitoring system will work. Another possibility is that PDPs themselves are able to set monitoring policies, with the particularity that the events are exclusively send to them. It is true as well that we can specify a mixed approach of both.
- *Monitoring Module Extensibility:* We expect each PDP has its own information model. But they may share a monitoring module. The reason to have a shared monitoring service is that - it is possible and adequate to abstract all those conditions we need to monitor, due to limited resources on a network element. The second reason is that monitoring is normally independent of policy semantics.
- *User, services and resources database info:* It might be interesting to have in the database of the element management system information regarding users, services and resources and relations between them. Moreover, it will be interesting that all PDPs within the PBANEM would be able to access this information.
- *Legacy routers support:* Should the Interface include methods for legacy routers, i.e. SNMP Support, MIB support, etc?
- *Dynamic Policy enforcement checking:* Before the policy reaches the PEPs, the PBANEM must be sure that the policy will be enforced. The PEPs can return a message for verification purposes.
- *Security and Credential Checking system*

3 R13 - ACTIVE SERVICE PROVISONING

3.1 INTRODUCTION

Telecommunications technology is rapidly evolving and new protocols and services are being developed, to satisfy the increasing user demands. However the deployment of these new protocols in the network is not a trivial task, as the nature of the Internet does not encourage the introduction of new services. The global network spans across multiple domains and includes thousands of routers. The deployment of a new protocol would require the software update in all these routers worldwide. As a result, although new protocols, such as IPv6 are being developed, the difficulty in their installation in the existing network infrastructure prevents their usage network wide.

The introduction of active networking aims to encourage the dynamic deployment of new network services. Active nodes provide a dynamic infrastructure that enables the introduction of new services network wide, while minimizing the overhead required. The FAIN project aims to develop such a flexible active infrastructure and to provide a mechanism for the dynamic installation of new network services.

In this context, the task of Active Service Provisioning (ASP) in FAIN has the objective to develop a framework for the distribution and installation of active services in the network. Taking advantage of the FAIN active infrastructure, the ASP architecture will demonstrate the ability to introduce new network services by the users of the active network. The users of the active network will be able to inject application-specific code, in order to provide-application specific network services The term user does not necessarily imply the end-user, but any user specified in the FAIN Enterprise Model, for example it can also include a service provider, who uses the active node infrastructure to build his own services and provide them to the end-user.

The FAIN ASP system will deal with the provision of service code components, as well as the provision of active code modules in general. The ASP infrastructure will also be responsible for the dynamic installation of new components of the FAIN Policy Based Network Management (PBNM) system, for example the PBNM system will use the ASP to dynamically install the code for a new Policy Enforcement Point.

The goal is to provide a single ASP architecture, which will be able to operate on all different types of active nodes that will be developed in FAIN (high-performance, DPE-based, Mobile Agent-based). The ASP architecture should also be generic and independent of the services that are being provisioned.

One of the goals of FAIN is to provide an active infrastructure, which enables the management of the network to be done on a per-customer and per-service basis. The Resource Control Framework (RCF) of the active node provides the ability to partition node resources in Virtual Environments (VEs), where the active services are instantiated. The FAIN Policy Based Network Management system also provides the opportunity to delegate management facilities to the user of the active network and uses the RCF to configure the resources of the node. For this reason, in order to effectively provide new services and to have the ability to configure the network on a per-service basis, it is important that the ASP system co-operates with the PBNM system.

In this chapter we identify the generic requirements for the provision of active services and active code, in general. We also make an overall description of the interactions required in the FAIN active network, in order to dynamically provide active services and to give the service (or its user) the ability to configure itself using service-specific policies.

In section 3.3, after the requirements analysis, we propose an architecture for the provision of active services. The components of the architecture are described and the corresponding interfaces are given.

In section 3.4 we describe an information model that will be used by the ASP system.

In section 5.5 we describe the requirements of ASP from other FAIN sub-systems, mainly the PBNM system, the security framework and the active node API. Wherever possible we provide some interface functions required by the ASP.

In section 3.6 we provide some overall scenarios covering the interactions inside the FAIN architecture required for the dynamic provision of a service and we describe the involvement of the different ASP components.

Section 3.7 contains the mapping of the ASP mechanisms to the FAIN Enterprise Model..

Finally, we mention some issues regarding ASP, which should be taken into consideration in the implementation phase and will likely be further investigated in the next two years.

3.2 REQUIREMENTS ANALYSIS

3.2.1 General Issues of Requirement Analysis

The aim of this section is to find and define high-level requirements for ASP. Basically we understand ASP framework as collection of mechanisms and services, which will enable users to provide new services or protocols in the Active Network (AN).

In the document we will use the term *service* instead of the protocol but do not make any distinction in between them.

High-level requirements for ASP are as follows:

1. It must support on the fly provisioning ofan arbitrary services in Active Nodes[18]
   - Service provisioning means that we can provide, remove, replace or modify a service on an ANN.
   - Arbitrary serviceand multiple different services should be supported and coexist in ANN.
   - Service provisioning must support all planes, e.g. user plane, control plane and management plane.
2. Request for service provisioning must be recognized and either realised or rejected for specific reasons.

   Requests can be explicit or implicit:
   - Explicit request: user[19] can request or provide a service[20]
   - Implicit request: a service can be provisioned because of the network conditions, etc.[21]
3. ASP should be independent of the type of the service.
   - There should be no difference between the provisioning of different services, for example between Reliable Multicast and VPN service provisioning.[22]
4. ASP should be interference free.

[18] See FAIN TA, [p.75].

[19] User: all possible users, defined in FAIN enterprise model, end user, (active) network operator, service provider, solution provider, management service provider, ...

[20] By an explicit request in a packet, or through an out-of-band request by the management system.

[21] For example in the case of reliable multicast; if the node sees too many NACKs in the return path to the source, specific code is run to enable NACK suppression and content caching. More examples are also possible, e.g. VPNs (ABLE demo), detection of the IPv6 packets and automatic tunnel setup, ETH media shaping demo ...

[22] We don't think to be wise to differentiate services in the way they are provisioned. Common mechanism should be provided for all active code (AC); functionality that enables AC to get executed and access resources on the node and in the network should be the same for all AC. Of course different services can be installed on the ANN in different ways.

- The provisioning of any single service should not affect the correct/secure operation of other services.[23]

5. ASP should be supported by arbitrary sets of active nodes.

-
- The constraints are a matter of policy not a matter of the system.

6. ASP should provide dependency resolution when necessary.

- Some services will have specific requirements that will have to be fulfilled to run the service. Some services depend on several distinct protocols, which by themselves do not provide complete services. Some services will depend on suitable services available in the environment for execution.

7. ASP should provide service discovery mechanisms.[24]

- In order to enable a user to choose an appropriate service (protocol) for their application, a service discovery mechanism should be provided.
- Service discovery mechanisms must support fully automated procedures without user intervention.

8. Services must be controlled for their resource usage.

- Needed resources should be given explicitly or they are extracted implicitly from the request.[25]
- The availability of needed resources should be checked upon request.
- Control for local and network wide usage of resources should be provided.

9. Mechanisms should be available for composing several services to provide composite services.[26]

- New, complex services can be composed by combining several independent services, which by themselves already provide complete services. In general it requires interaction among services.

10. ASP must be secure[27]

- Secure provisioning means that we know where we get code from, the code has been transferred securely and request for provisioning has to be authorized regarding the node/environment policy.
- Code has to be verified and policy enforced, so that the code execution and operation has to be monitored. If requested, code policies could be inserted in other policies database.
- Code communication and message exchange with other code should be restricted and monitored to be in line with the node/environment/other-code policies.
- Access to communication data (packets) should be checked and secure code termination and removal from the node should be possible.

---

[23] In this requirement we do not initially consider interference while code is running on the node; we have limited our scope only to service initialization. But there can be also interference during code operation (running on the node); for example different or the same services requested by different users which compete for scarce node resources or limited resources available to one user.

[24] Mainly for out-of-band code. In-band code is transparent in this regard when thinking where/how to find code.

[25] It can be seen also as dependency information, e.g. a service needs some resources and therefore cannot be run on this node, but maybe it can be can run on some other node.

[26] Note that this is an optional requirement, which will not be extensively covered at this stage, it is mentioned for the sake of completeness

[27] For more broad discussion see [1] and [2].

11. Interoperability between different nodes and different environments for execution should be provided.
    - Interoperability between different nodes, e.g. two differently designed and built ANN implementing the same service should interoperate in the context of the service.
    - Interoperability between different environments for execution, e.g. two different environments hosting/running the same service should interoperate in the service's context.
    - Interoperability between different implementation of a service, e.g. two different implementations of the same service should interoperate.
12. It must be possible to disable on the fly operation of certain active code.[28]
13. Upgrading or replacing existing services should be dynamic.[29]
14. The end-to-end communication should not be interrupted when a protocol or service is removed (replaced, enhanced).[30]

3.2.2 Service Independent ASP Use Cases

This section attempts to identify a number of service independent use cases concerning Dynamic Protocol/Service Provision. The procedures followed for this identification is based on the use case identification process used for the applications in FAIN WP2. We have to mention that this list is not complete and the use cases mentioned here are not specified in full detail. The main objective of these use cases is not to describe the overall ASP architecture, but mostly to serve as a basic requirement analysis for the system.

In the following use cases we describe the generic interactions taking place for the provision of an active service. We are not restricted to the process of downloading service code to the active node, but we also take in mind issues about service configuration and resource management and allocation on a per-application basis. Some of these will not be implemented inside the FAIN ASP system, but they will be parts of the PBNM system and the node RCF architecture. We have chosen to include all these processes in our use cases, in order to provide a more complete view of the overall system interactions required for the provision of a service. To avoid confusion, we explicitly state which mechanisms will not be implemented inside the ASP system.

3.2.2.1 Service independent ASP use cases hierarchy

In the figures below, five sets of Service Independent use cases for the ASP are given:

[28] Has much to do with revocation, not only of the running code but also of the cached code, i.e.there should exist mechanisms that should allow to revoke code from the network. On the other hand, the code revocation should not interrupt end host to end host communication..

[29] See for example [3].

[30] This is a curcial point if f we want a realistic and scalable AN approach where AN applications should not break end-to-end connectivity but help end-to-end clients to communicate in more efficient way. If services or protocols are not available, end clients should communicate over their own protocol stacks (active/passive) and without additional active network services.

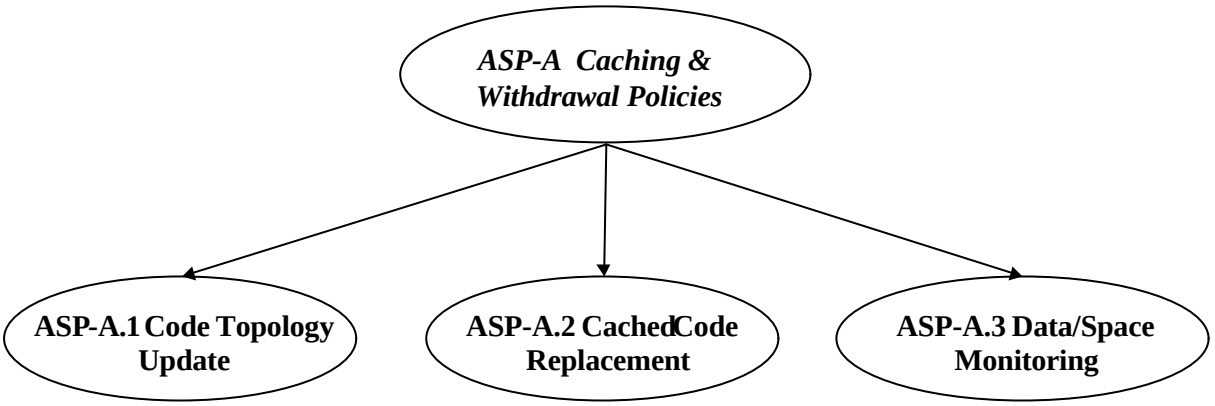

Figure 46 – Set A (Caching & Withdrawal Policies)

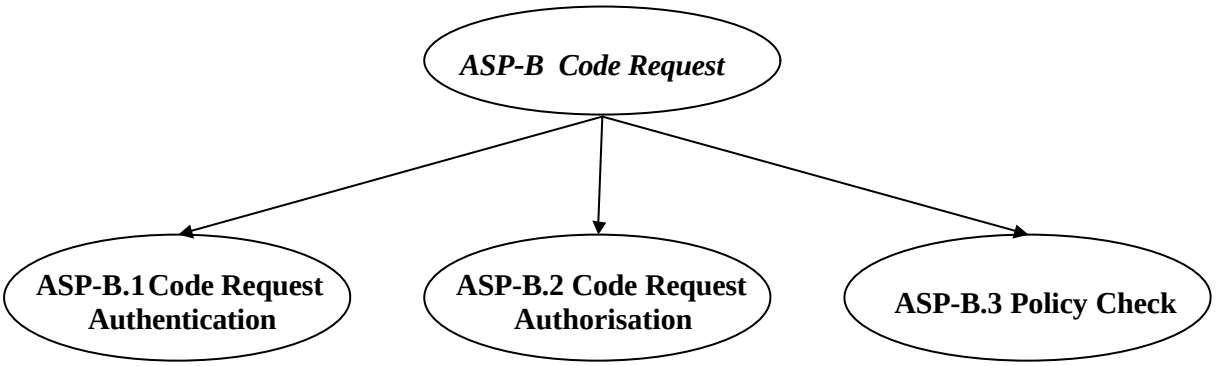

Figure 47 – Set B (Code Request)

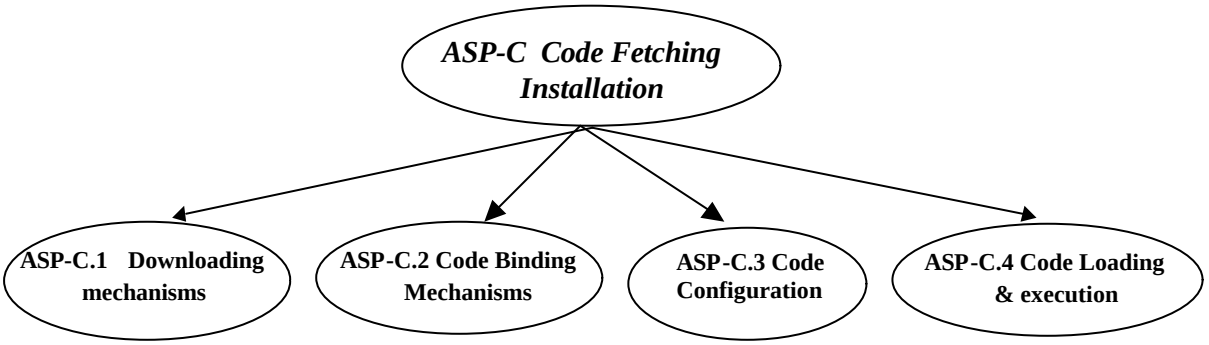

Figure 48 – Set C (Code Fetching Installation)

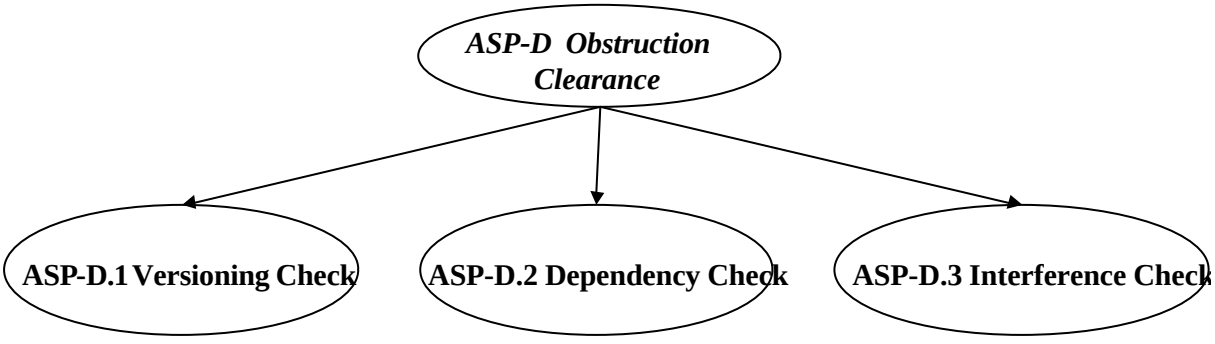

Figure 49 – Set D (Obstruction Clearance)

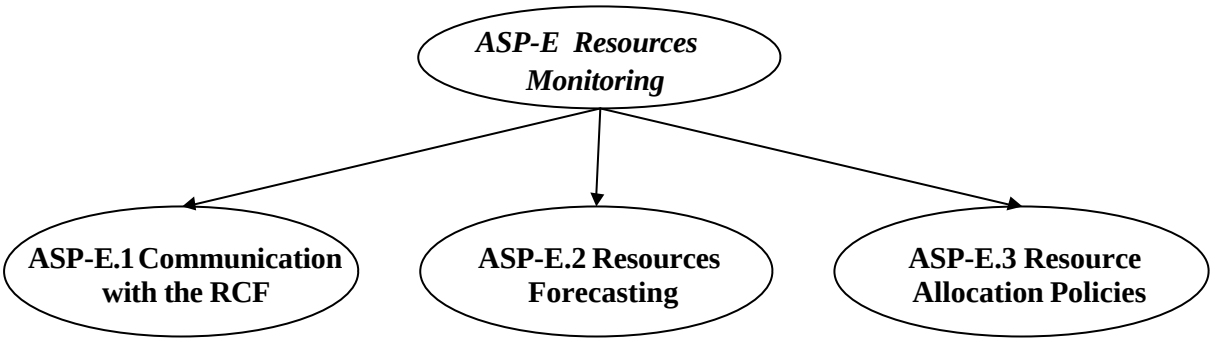

Figure 50 – Set E (Resources Monitoring)

3.2.2.2 Service independent ASP use cases description

This Section discusses a number of use cases that are not bound to specific applications but rather describe the generic functionality that the ASP Architecture should provide. For categorisation purposes we have use the following numbering: *ASP-X.No.No* where *X* is a letter for A to Z and the *Nos* are a number starting from 1. In this way the use case hierarchy is shown.

ASP-A ***Caching & withdrawal Policies***. This set of use cases covers all the operations which need to be carried out by the ASP, either during code provisioning or in a synchronous manner for locating and storing existing code in an efficient and rapid way.

ASP-A.1 **"Code Topology" update**. After a service/protocol withdrawal either in the node or in the neighbouring nodes a code registry facility of the ASP should be updated. Alternatively, this update could be either a synchronous event or a reactive action (after a code fetching false from the code repository).

ASP-A.2 **Cached Code Replacement**. The processes and the policies for code replacement either in the main memory of the system, in dedicated buffers, in possible caches or processor specific memory (in the case of multiprocessors and/or multilayer memory hierarchy network nodes) are described in this use case.

ASP-A.3 **Data/Space Monitoring**. This use case is strongly related with ASP-A.1 since it describes the way that the ASP should monitor free space availability on the system for code caching before applying and code topology update or code fetching.

ASP-B ***Code Request***. This set of use cases considers the preliminary operations that should be carried out before the actual operation of code fetching will be performed. These operations mostly concern security issues of the request. These issues will be handled by the FAIN PBNM system and the security architecture. First, the necessary checks will be performed and then the request will be delivered to the ASP system

ASP-B.1 **Code Request Authentication**. Users requesting service provisioning access must be authenticated to the FAIN security framework.

ASP-B.2 **Code Request Authorisation**. Upon user authentication and credentials provided, user request for code provisioning must be authorized by security subsystem.

ASP-B.3 **Policy Check**. Apart from the typical authorisation/authentication a number of policy enforcement mechanisms should exist during code request resolving. These policies will check the potential capabilities of the system to host the new request and will possibly prevent congested situations.

ASP-C ***Code fetching/installing***. This set of use cases considers the main mechanisms of downloading, binding and executing the provided code. The ASP system is focused not only on the download of code to the node, but also with other tasks like checking code dependencies, versioning and interference during service provisioning, while the actual injection and execution will be done by the NodeOS.

ASP-C.1 Downloading Mechanism. The operation of downloading code to the Node is a major task for code provisioning. The ASP system downloads the requested active code module from the network and passes it to the active network node, where it will be injected in the appropriate VE.

ASP-C.2 Code Binding Mechanisms. Code binding is not described here, it is just mentioned for stressing the fact that ASP is strongly related with the main operations of the NodeOS

ASP-C.3 Code Configuration. After service code is downloaded to corresponding nodes and interfaces on which code execution depends upon are bound, often additional configuration of parameters required before execution of service code is required. Please note that configuration of service code can take place at every stage of active service provision: before downloading, execution and during execution of service code.

ASP-C.4 Code loading & execution. The operation of code loading and execution marginally belongs to this set of use cases. This is due to the fact that the loading and execution of the code is considered as a main task of the core NodeOS and/or the EE, therefore it is for further study from the related task groups of FAIN. Here is mentioned just for the sake of completeness.

The actual responsibility of the ASP system in this use case is to deliver the code module to the NodeOS, using the interface offered by the active node.

ASP-D ***Obstruction clearance***. By the term "Obstruction" we consider a number of difficulties of that could be faced during code provisioning.

ASP-D.1 Versioning Check. Describes the necessary checks that should be carried out for determining whether or not the requested code is up to date or an updated version is needed. This issue could be considered essential in some cases (e.g. evolving protocols and services). We can expect that multiple versions of the same code can be run on the system if this is needed for backward compatibility and is explicitly allowed by the security policy.

ASP-D.2 Dependency Check. It is certain that for a (large) number of code provisioning cases there will be code dependencies from other software modules (e.g. a protocol dependency). This means that the whole code provisioning process will have to be carried out again (and again). Even though this is unavoidable we should be very careful in resolving these dependencies and we should clarify the scenarios where we may act proactively, e.g. by checking code dependencies before initial code downloading or by downloading the complete code stack with minimal checking.

ASP-D.3 Interference resolution. Considers interference in terms of communication and computational resources and is therefore strongly related with the Resource Control Framework and management subsystem. We can consider three types of interference; the first category is about the distribution of limited resources to a set of requests stemming from the execution of the provisioned code. The second category considers resources that can process one request at a time (e.g. a video transcoder). The estimation of resources and the definition of resource utilisation policies are considered essential for the efficient operation of the whole node. The third type is interferences checked by the security framework, e.g. checking user available resources.

The FAIN PBNM system, in cooperation with the node Resource Control Framework will guarantee that each service running in the active node will be able to reserve its own resources, which will not be accessible by other services. The security framework will take care of security issues. The ASP system will not have to do any additional work to prevent interference between different code modules.

ASP-E ***Resource Monitoring***. For the proper functioning of most of the operations mentioned above an adequate resource monitoring facility should exist. There should exist the ability to allocate resources to the code, which is installed in an active node, and to monitor the resource consumption of active applications, to ensure that the agreed limits have not been exceeded. Information about allowed resource usage is a matter of the supplied credentials and security policy. The security framework is responsible for enforcement decision during service provisioning while the FAIN PBNM system is responsible for the enforcement of these requirements during the service operation, in co-operation with the Resource Control Framework of the active node. If resource conflicts or violations exist during service provisioning, the security framework is responsible for cancelling the operation. The ASP system does not have to deal directly with these issues, so we initially omit the details of these three individual use cases.

3.2.2.3 Tabular description of ASP use cases

In this section, the ASP use cases, briefly introduced in the previous section are presented. To be consistent with the use case description process followed in WP2 we use the same tabular template.

Use Case Name:	**ASP-A Caching Policy**
Summary:	This set of use cases covers all the operations, which need to be carried out by the ASP either during code provisioning or synchronously for locating and storing existing code in an efficient and rapid way.
Basic Course of Events:	The Course of events are described at the “child” use-cases (ASP-A.1 to ASP-1.3)
Alternative Paths:	Possible bypass of caching mechanisms for simplifying the ASP Architecture.
Extension Points:	N/A
Trigger:	N/A
Assumption:	
Pre-condition:	N/A
Post-condition:	N/A
Remark:	

Use Case Name:	**ASP-A.1 “Code Topology” update**
Summary:	After a service/protocol withdrawal either in the node or in the neighbouring nodes the code directory service of the ASP should be updated. Alternatively, this update could be either a synchronous event or a reactive action (after a code fetching false) to maintain a consistent view of the services/protocols installed in the network.

	This information could be used for faster retrieval of active code.
Basic Course of Events:	1. Look-up the contents of the code stack 2. Look-up the contents of the code repository 3. Look-up request to the boundary Active Nodes
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	• Asynchronously after code (service/protocol) withdrawal • Asynchronously after a code fetching phase • Synchronously, in a predefined time or after a certain time period
Assumption:	This use case assumes three types of code storage: 1. *Code Stack*, where the contents of the stack are ready for execution 2. *Node Repository*, where the code is available but needs to be fetched into the Code stack. 3. *Network Repository*, There should be a central code repository in the network. Alternatively, code stored at the repository of the boundary nodes could be used as a distributed repository by the node.
Pre-condition:	• Existence of a data/space monitoring mechanism. • Existence of Caching policies
Post-condition:	N/A
Remark:	Step 3 assumes that each node should provide the necessary information. In a later stage we might need to describe this mechanism as an additional use case.

Use Case Name:	**ASP-A.2 Cached Code Replacement**
Summary:	The processes and the policies for code replacement either in the main memory of the system, in dedicated buffers, in possible caches or in processor specific memory (in the case of multiprocessors and/or multilayer memory hierarchy network nodes)
Basic Course of Events:	1. A request for code installation arrives 2. The required free space in the code repository are computed 3. Free space is resolved (see also Use Case ASP-A.3) 4. In case of space unavailability a code replacement algorithm is performed 5. A “ready for installation” message is produced. 6. Resolution of free space in the code stack is performed and code replacement algorithm is also performed

	7. After code uploading to the code repository, the code is copied from the code repository to the code stack and is ready for execution 8. A request for code topology update is posted (see use case ASP-A.1)
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	• Use-case ASP-A.1 • Upon code request when the code in unavailable in the node's code buffers
Assumption:	• See assumptions in ASP-A.1 • In steps 4 and 6, examples of the predefined code replacement algorithms could be FIFO, LRU, random etc.
Pre-condition:	N/A
Post-condition:	N/A
Remark:	Due to a current lack of concrete node memory architecture, this use-case will likely need further refinement

Use Case Name:	**ASP-A.3 Data/Space Monitoring**
Summary:	This use case is strongly related with ASP-A.1 since it describes the way that the ASP should monitor free space availability on the system for code caching before applying code topology updates or code fetching.
Basic Course of Events:	1. Fetch code that should be stored 2. Apply the replacement policy 3. Store new code 4. Update the code registry
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	• Use-Case ASP-A.1 • Asynchronously after fetching a new service/protocol in the node • Synchronously after a predefined period
Assumption:	
Pre-condition:	N/A
Post-condition:	N/A
Remark:	

Use Case Name:	**ASP-B Code Request**
Summary:	This set of use cases considers the preliminary operations that should be carried out before the actual operation of code fetching will be performed. These operations mostly concern security issues of the request. The necessary checks will have to be performed by the PBNM system and the FAIN security framework. The ASP will assume that the request has already been authorised and all necessary resources have been reserved.
Basic Course of Events:	1. Resolution of the incoming service/protocol request in terms of requested code needed. 2. Authentication, authorisation, policy checks are done by the security framework and the PBNM system. 3. Request transferred to the ASP system. 4. Code Availability check is carried out 5. Dependency Check is carried out (see Use case ASP-D.2) 6. Clearance for Service Execution is given
Alternative Paths:	In the case that after the dependency check an additional service/protocol is required, the necessary security and policy checks should also be made for the new code.
Extension Points:	N/A
Trigger:	• An incoming service/protocol request
Assumption:	
Pre-condition:	N/A
Post-condition:	N/A
Remark:	

Use Case Name:	**ASP-C Code fetching/installing**
Summary:	This set of use cases considered the main mechanisms of downloading, binding and executing the provided code. The ASP system will locate the requested code in the network, download it and deliver it to the NodeOS/EE for the actual installation.
Basic Course of Events:	The course of events is described in the following use cases
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	N/A
Assumption:	
Pre-condition:	N/A
Post-condition:	N/A
Remark:	

Use Case Name:	**ASP-C.1 Downloading Mechanism**
Summary:	The operation of downloading code to the Node is a major task for code provision.
Basic Course of Events:	1. The ASP system first checks a local code cache 2. If the code is not stored locally, the ASP searches the network. It can either lookup in a neighbouring node, or in a well-known code server. 3. A security mechanism in the code server should authenticate the communication. 4. Code is downloaded to the node 5. Necessary checks have to be done by the security framework. 6. Code is delivered to the NodeOS loader, to be injected in the appropriate Execution Environment.
Alternative Paths:	If the code is stored in the local cache, it is installed immediately without searching in the network.
Extension Points:	N/A
Trigger:	N/A
Assumption:	In the simplest case, we have to assume that code cache is code/service/user dependent. If this is not a case, and other user code in cache is used, all necessary security checks have to be performed. In this case there is also a need for a smart cache manager to not reuse the code, which is unsuitable for the current operation. For example the same code with wrong digital signature.
Pre-condition:	N/A
Post-condition:	N/A
Remark:	

Use Case Name:	**ASP-C.2 Code Binding Mechanisms**
Summary:	Code binding is not described here, it is just mentioned for stressing the fact that ASP is strongly related with the main operations of the NodeOS and therefore a number of interfaces and boundary operations (such as code binding) should be addressed and designed in an early stage.
Basic Course of Events:	The implementation of the code binding mechanisms is not a responsibility of ASP.
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	N/A
Assumption:	

Pre-condition:	N/A
Post-condition:	N/A
Remark:	

Use Case Name:	**ASP-C.3 Code Configuration**
Summary:	After service code is downloaded to the corresponding nodes and interfaces on which code execution depends upon are bound, often additional configuration of parameters is required before the execution of service code. Please note that configuration of service code can take place at every stage of active service provision: before downloading, execution and during execution of service code.
Basic Course of Events:	Not a responsibility of ASP. The service will interact with the management system in order to do the necessary configuration.
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	N/A
Assumption:	
Pre-condition:	N/A
Post-condition:	N/A
Remark:	

Use Case Name:	**ASP-C.4 Code loading & execution**
Summary:	The operation of code loading and execution marginally belongs to this set of use cases, since loading and execution of the code is considered as a main task of the core NodeOS and/or the EE. As such, it is for further study from the related task groups of FAIN. Here it is mentioned just for the sake of completeness.
Basic Course of Events:	To be done by WP3. Implementation is specific to the type of Execution Environment that will exist in the node.
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	N/A
Assumption:	
Pre-condition:	N/A
Post-condition:	N/A
Remark:	

Use Case Name:	**ASP-D Obstruction clearance**
Summary:	By the term "Obstruction" we consider a number of difficulties that could be faced during code provision to impede the proper operation
Basic Course of Events:	This use case is a set of diverse use cases therefore there is no general course of events.
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	N/A
Assumption:	
Pre-condition:	N/A
Post-condition:	N/A
Remark:	

Use Case Name:	**ASP-D.1 Versioning Check**
Summary:	Describes the necessary checks that should be carried out for determining whether or not the requested code is up to date or a version update is needed. This issue could be considered essential in some cases (e.g. evolving protocols and services)
Basic Course of Events:	1. The version of the requested service/protocol is resolved 2. During code availability look-up at the code registry a version cross check is performed 3. In case of version incompatibility (previous version) the code is considered unavailable and the code fetching procedure should be performed.
Alternative Paths:	If a previous version of the service is required for some service operation, the new environment with new code should be started if this operation is explicitly allowed by security policies.
Extension Points:	N/A
Trigger:	N/A
Assumption:	
Pre-condition:	N/A
Post-condition:	N/A
Remark:	The versioning compatibility is a major issue since we consider that only one version of the code should exist on the node. This means that: • Backwards compatibility is ESSENTIAL for supporting all running applications in the node. • In case of versioning update, older applications should be reconfigured to use the new code (e.g. a protocol update)

Use Case Name:	**ASP-D.2 Dependency Check**
Summary:	It is certain that for a (large) number of code provisioning there will be code dependencies from other software modules (e.g. a protocol dependency). This means that the whole code provisioning process will have to be carried out repeatedly. Even though this is unavoidable we should be very careful in resolving these dependencies and we should clarify the scenarios where we may act proactively, by either checking code dependencies before initial code downloading or by downloading the complete code stack before checking.
Basic Course of Events:	1. A specific service request is given as an input 2. Look-up the dependency list to resolve possible code dependencies 3. Look-up for possible additional necessary code in both the code stack and/or code repository 4. Return the results in code dependencies (both available and unavailable in the node)
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	Step 5 of Use Case ASP-B
Assumption:	A Dependency list is a required facility for this mechanism. The update of this list follows the code update mechanisms
Pre-condition:	N/A
Post-condition:	N/A
Remark:	In this use case we do not foresee any information about code dependencies coming with the service request, this makes the service request simple and general but of course increases the complexity of ASP architecture. It is under discussion whether or not we keep this approach or we skip out the dependency check.

Use Case Name:	**ASP-D.3 Interference resolution**
Summary:	Considers interference in terms of communication and computational resources therefore is strongly related with the Resource Control Framework. We can consider three types of interference; the first category is about the distribution of limited resources to a set of requests stemming from the execution of the provisioned code. The second category considers resources that can process one request at a time (e.g. a video transcoder). The third category is interferences checked by the security framework, e.g. checking users available resources. The estimation of resources and the definition of resource utilisation

	policies are considered essential for the efficient operation of the whole node.
Basic Course of Events:	1. Estimation of the resources needed for the execution of the service instance 2. Security check for allowed resource usage. 3. Possible insertion of resource requests to the system (process queues, registers etc.) 4. Triggering mutex mechanisms for specific resources (e.g. transcoders, dedicated processors etc.) 5. Set the service to running or to waiting state
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	From Use Case ASP-C, before code execution
Assumption:	
Pre-condition:	N/A
Post-condition:	N/A
Remark:	There is always the possibility of not having adequate resources for serving the request. In this case we may having a Denial of Service (DoS) or delays in service execution (based on the QoS requirements from the service) This use case will not be implemented by the ASP system, but by the node Resource Control Framework.

Use Case Name:	**ASP-E Resource Monitoring**
Summary:	For the proper operation of most of the activities mentioned above, an adequate resource monitoring facility should exist. The interactions with the Resource Control Framework that will be specified as a part of node infrastructure, resource allocation policies need to be defined and the existence of resource-forecasting mechanisms, will be useful in the improvement of the overall system. To avoid inserting redundant functionality in the ASP system, these tasks will be handled exclusively by the management system
Basic Course of Events:	In order to avoid having duplicate functionality inside FAIN, the management system will configure and monitor the resources of the node, using the Resource Control Framework.
Alternative Paths:	N/A
Extension Points:	N/A
Trigger:	N/A
Assumption:	
Pre-condition:	N/A

Post-condition:	N/A
Remark:	

3.3 ASP ARCHITECTURE

This section describes the Active Service Provisioning Architecture (ASP). ASP is closely related to the management system and comprises the installation, (re)configuration and removal of services in an active network environment. Part of the functionality described in this section may be implemented in the management system. Nevertheless it is included here to show the complete ASP functionality and to ease the understanding of the ASP architecture.

3.3.1 Services

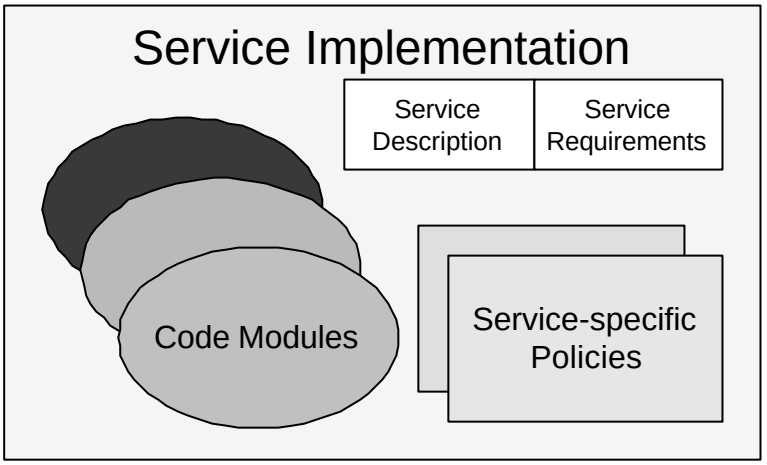

Figure 51 - Service Implementation

A service implementation consists of: a service description, service requirements, service code modules, and service-specific policies. These service components are described in the following paragraphs.

- *Service description* specifies the functionality provided by the service as well as its version (cf. use case ASP-D.1). This information is entered into a database (the service registry). It is used for service discovery.
- *Service requirements* describe the Virtual Environment (VE) that the service has been written for. Implicitly, this specifies the API that is required by the code modules. If the specification of a VE allows for optional functionality, the requirements also list which kind of VE functionality is mandatory for proper service execution. E.g. if IPsec functionality is optional for a certain VE, but the service relies on it, then the service requirements must state this fact. In this way dependency of a service implementation on the underlying software/hardware environment is resolved (cf. use case ASP-D.2).
- *Service-specific policies* describe the (re)configuration of the active network environment that a service requires for proper execution. Configuration of the active network environment includes the reservation of resources as provided by the resource control framework (RCF). Please note that there is no direct interface between a service and the RCF. In fact, communication between service and RCF is done via installation of service-specific policies in the management system. The management system decides on whether to deny or accept the installation of such policies with the help of the security framework. Moreover, policies allow not only (re)configuration of the environment, but also of the service itself. This is an important property and allows taking full

advantage of the active network concept. Depending on service-specific policies, a service may request the installation/removal of service code modules at runtime.

- *Code modules* are service components that are supposed to be executed on the network nodes. The FAIN cube model allows for the execution of code on the data path, control, and management plane. Depending on the type of VE/EE, these modules may implement functionality on one or more planes. E.g. a code module for the data plane could implement a transcoding function whereas a code module for the management plane might implement a Policy Enforcement Point (PEP) that is required by a service-specific policy.

3.3.2 ASP Network Architecture

This section describes the functionality of ASP with a network wide scope. The network wide ASP architecture is presented in the following figure.

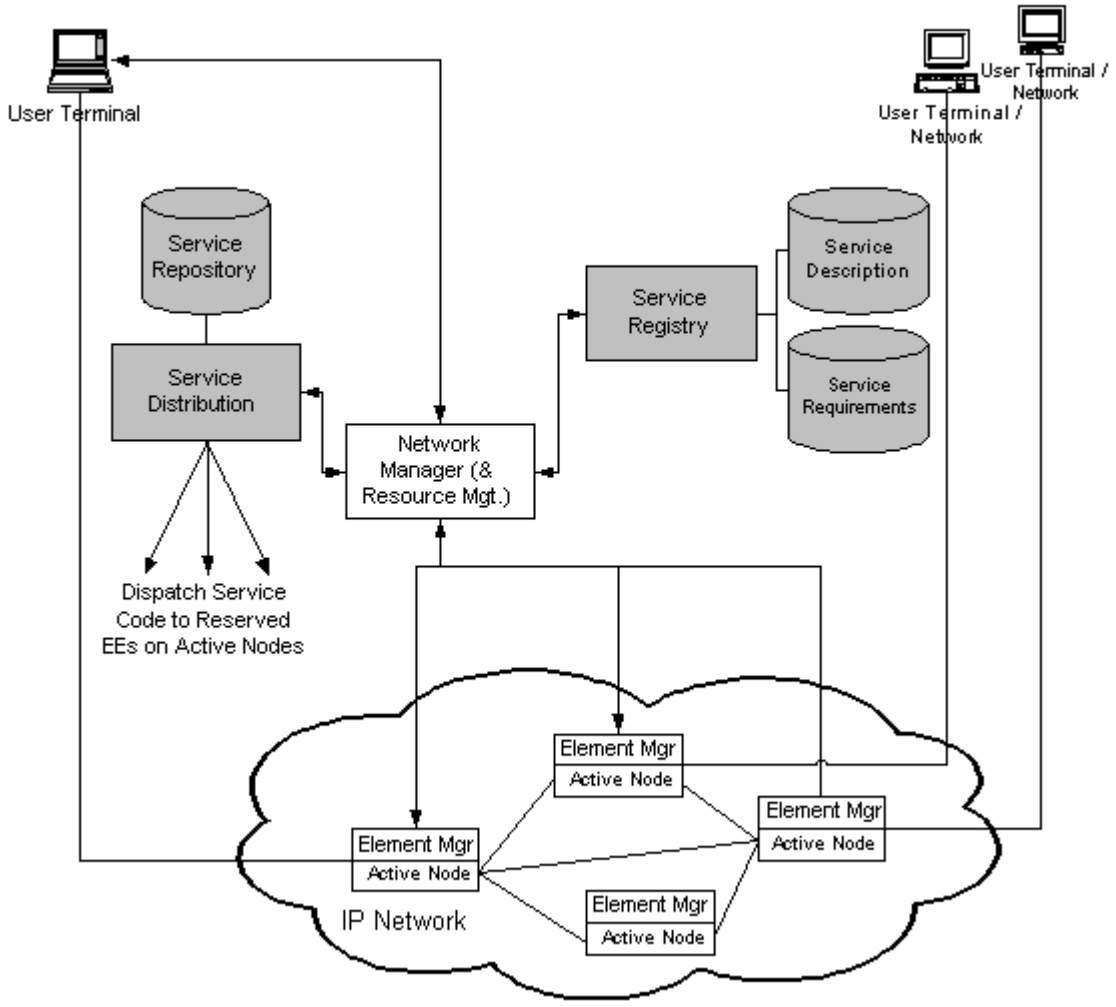

Figure 52 - Active Service Provisioning Architecture

In the network level the ASP contains the code repository, where the active code modules, which can be dynamically installed in the nodes, are stored. A service registry contains information about the available services and about their mapping to specific implementations. For this reason the registry should maintain information about the requirements of each service. In addition, the service registry is aware of the location of the code repository, where the implementation of the corresponding service is stored. A service distribution component provides the ability to initiate the network-wide distribution of a service in a group of nodes.

3.3.2.1 Service Repository

The repository contains the code modules that implement the active services offered in the network. Active code can have various formats, depending upon the implementation of the Execution Environment where the code can run. For example active code for an EE based on the Java virtual machine can be stored as the corresponding Java class file (as in ANTS). The service distribution component can access the repository, to retrieve the necessary code modules and download them to specific active nodes.

3.3.2.1.1 Interface functions

For the retrieval of the code modules the repository should support the following functions

getCodeModule

Retrieves the specified code module from the repository

in: CodeId

out: Code

storeCodeModule

Stores a code module in the repository

in: CodeId

in: Code

deleteCodeModule

Deletes a code module from the repository

in: CodeId

3.3.2.2 Service Registry

The availability of a service implementation must be made known to the network users. Therefore one or more service registries exist within the network. The service registry acts as a name server that maps the (functionality) description of a service to one or more service implementations and their respective requirements. The requirements of the service can include a specific type of node or EE, dependencies on other services and on specific implementation versions. Moreover the service registry is aware of the location of the code server(s) that contain the service code modules and the service-specific policies.

3.3.2.2.1 Interface functions

The following component interface functions are proposed:

registerService

registers a service in the service registry and stores the code modules and service-specific policies in the specified repositories.

in: service_description

in: service_repository_locations

in: service_properties

in: service_implementation

findService

returns a list of service repositories where services matching the service description (functionality) can be found.

in: service_description

in: service_properties (optional)

out: service_list

unregisterService

deletes the entry for a service from the service registry and removes the service components from the service repositories.

in: service_description

in: service_properties

serviceMap

used by the management system, to obtain information about the implementation of a service and its requirements

in: ServiceId

in: Credential

Information regarding the principal who made the request

in:ReqResources

Information regarding the principal associated with the principal who made the request

out: CodeModulesInfo

Contains information regarding: the code module(s) that should be installed, the type of ANN and the VE where it should be installed, dependencies with other code modules and optionally the minimum recommended resources for that code module

3.3.2.3 Service Distribution

Service distribution deals with the deployment of service code modules in the network. Therefore the service components need to be encapsulated and shipped to the appropriate nodes. Different technologies exist for this purpose. Some approaches (StreamCode [102]) make use of active packets, where executable code and data are contained in the same packet. Another possibility ANN[101] is to embed in a packet a reference to a code module. On receiving such a packet, an active node retrieves the module from the local cache or – if necessary - downloads the module from a code server. It is also possible to embed service components in mobile agents visiting the nodes and requesting the installation of the service components. Finally, it has been proposed to use a Distributed Processing Environment (DPE), relying on a CORBA-like infrastructure to deploy the service.

3.3.2.3.1 Interface functions

The following component interface functions are proposed:

distributeService

Initiate service distribution. A service_provision_session_id is generated. Distribution_qos is either "best-effort" or "all-or-none".

in: service_description

in: target_nodes

in: distribution_qos

out: service_provision_session_id

getServiceDistributionState

returns the state of service distribution (di, installed)

in: service_provision_session_id

out: service_distribution_state

3.3.2.4 Network Manager

The Network Manager acts as a central component in the ASP architecture. It communicates directly with all other components of the ASP architecture. The network manager has been specified in chapter 2. Therefore, we restrict the description of the network manager to the ASP related issues.

From an ASP viewpoint, the network manager provides the following functionality.

- Process service requests from users
- Get service offers from one or more service registries
- Install service-specific policies (includes mapping the service to the nodes and resource allocation)
- Initiate service distribution
- Modify service state (run, terminate)

A service consists of executable code and service-specific policies describing the requirements of the service in order to run properly. After a service has been requested and the appropriate service has been discovered in the network, the network manager is required to install the network wide policies that come with the service. As a consequence, the Resource Control Framework will reserve network resources for this service. In case the installation of the policies fails - e.g. because the network manager discovers conflicts with other installed policies or the required resources cannot be allocated - the service request is rejected.

All requests have to be intercepted by the security framework subsystem and on the basis of the proper authentication and/or credentials granted or refused.

A service may reconfigure itself by requesting the installation/removal of service-specific policies on the network level. However, these actions can only be performed under the condition that all security framework checks are passed and that the management system did not detect any policy conflicts.

3.3.2.4.1 Interface functions

The network manager is not a part of the ASP architecture and for this reason we do not list here the corresponding interface. For more detail, the reader is referred to chapter 2.

3.3.3 ASP Node Architecture

In this section we present the ASP architecture at the node level in more detail. The purpose of the node-level ASP components is first of all to have a point in the node where the requests for installation of a service or a code module can be processed. Additionally, in order to improve overall performance the ASP should locally cache recently used code modules at the node, so that it will not be necessary for every code request to have to download the corresponding code module from the network code repository. In order to provide more distributed ASP functionality, we could also have a local registry, which will implement a sub-set of the network-level service registry.

The Node ASP architecture is described in the following figure

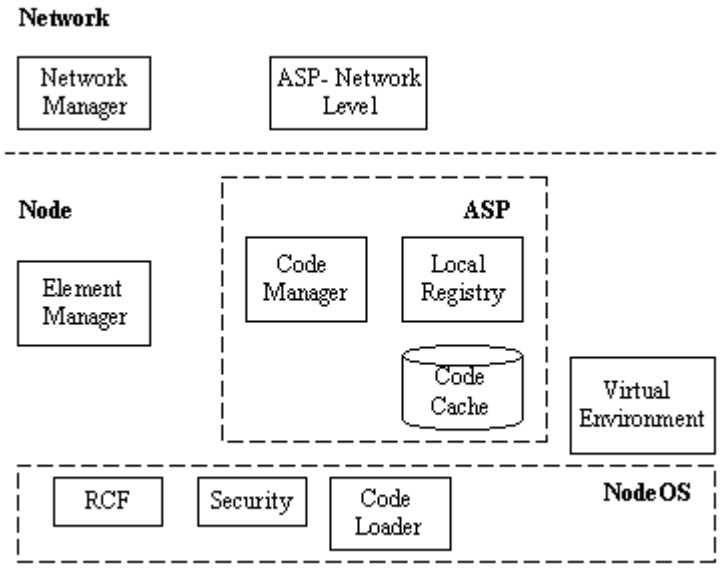

Figure 53 – ASP Node Architecture

3.3.3.1 Code Manager

The ASP code manager is the core of the architecture at the node level. The code manager acts as a request processor, which receives requests for code installation or removal either from the management system or from the service distribution network component. When the request is received from the management system, the ASP does not need to perform any additional checks for the authorisation of the user request and for the reservation of the necessary resources. The management system will already have processed the request and will pass the necessary information for user credentials and for the place where the code should be installed as parameters of the request.

Additionally, the code manager can also receive a request to install code to upgrade the functionality of the management system, for example it can install the code for a new Policy Enforcement Point.

The code manager can consult the local code cache to check if the requested code is already present on the node, or else it can lookup in the service registry and get access to the central code repository of the network.

The manipulation of active code is done by communicating with the node loader, which will be a part of the node operating system. For this reason, the node loader should support the appropriate methods to install or remove an active code module, or to change the status of a loaded code module (suspend, resume or stop it). When interacting with the node loader, the code manager has to provide the credentials of the principal who initiated the request, so that the necessary checks can be made inside the NodeOS.

3.3.3.1.1 Interface functions

downloadReq

A request for the installation of an active code module

in:CodeID

in:Credential

in:Pointer

The parameters that would be required for this method should carry information related with:

CodeID, which will be a string identifying the needed code package.

Credential: information regarding the principal associated with the principal who made the request.

Pointer: is a string that indicates to the ASP the place (e.g. a directory) where the package should be downloaded.

changeServiceCodeState

Change service code state to new state. State may take the following values: "running", "stopped", "suspended".

in: service_session_id

in: new_state

3.3.3.2 Code Cache

Service code may be cached on a node. We propose the use of a simple caching scheme. If code is to be installed on a node, the local cache is checked for the code modules. The implementation of a more complex caching scheme where the caches of other nodes are checked, are considered to be of interest and may be addressed in a later phase of the project.

The simplest caching algorithm is to store recently used files, based on their last access. Because code may be updated with newer versions, there could be an expiration timer for each active code module. When a code module is cached for more time than specified, it can be removed from the node and it will have to be downloaded by new from the code repository.

Due to security reasons, the code cache should be able to associate the cached code with the user, on behalf of whom the code was downloaded to the node. In the case that the same code is requested by another user, it will have to undergo the necessary security checks again, although it already exists in the node.

3.3.3.2.1 Interface functions

getModuleList

returns a list of code modules available in the local cache

out: module_list

getModule

checks whether the specified code module is stored in the cache. if this is the case, it returns a copy of the code module.

in: code_module_id

out: code_module

insertModule

stores a code module in the local cache

in: code_module_id

in: code_module

removeModule

removes code module from the local cache

in: code_module_id

in: code_module

3.3.3.3 Local Registry

The local registry component implements a sub-set of the network-level service registry at the node level of the ASP architecture, in order to provide the necessary information locally. The local registry may cache the descriptions and the requirements of recently accessed services, so that if a request comes from the element manager, it will be answered without having to access the central network registry. The same issues apply here as to the caching of active code, concerning the replacement or refresh of cached information.

Optionally, the local registry may contain information about code modules cached in neighbouring nodes, so that we implement a form of distributed caching to improve download performance. In this case it is necessary to have a mechanism for the exchange of this information between neighbouring nodes. This issue will be left for the next phase of the project.

3.3.3.3.1 Interface functions

The local registry will implement the following functions, specified for the service registry in the network-wide architecture.

serviceMap

registerService

unregisterService

Additional functions may be needed in order to perform distributed caching of code. If needed, they will be specified as work progresses in FAIN.

3.3.3.4 Element Manager

The Element Manager is mainly specified in chapter 2. Therefore, we restrict the description of the Element Manager functionality to the ASP related issues.

On a node local level, service-specific policies are needed to reserve node resources and to control/configure other relevant components of the node for the service to perform as intended. These requirements are described by policies that must be installed on the same node where the service code runs.

We note that a service code module may interact with the node resource control via the node element manager. Therefore a service code module should be able to dynamically install, modify and remove policies on the node. However, these actions can only be performed under the condition that security framework subsystem checks are passed and that management system did not detect any policy conflicts.

Service-specific policies should be able to replace service code modules based on local state.

3.3.3.4.1 Interface functions

As the Element Manager is not part of the ASP architecture, we do not describe here the corresponding interface. The interface that will be offered by the Element Manager to the ASP system is interface 1, as specified previously in chapter 2.

3.3.3.5 Code Loader

A code loader will handle executable code modules that were brought to the node. The code loader will be specified in the node architecture document. The following section therefore serves as requirements to the node architecture. This component allows code to be installed, started, suspended and removed. Starting the code involves allocating the appropriate node resources to the code, loading the code into the memory and executing it. Suspending code means blocking a certain process. If a code module is suspended, packets are dropped. Code is removed by stopping a process and freeing the node resources allocated to that process.

3.3.3.5.1 Interface functions

loadModule

loads a service code module into the appropriate execution environment and configures it.

in: service_session_id

in: code_module_id

in: code_configuration

removeModule

removes a service code module from the execution environment.

in: service_session_id

in: code_module_id

configureModule

(re)configures module on runtime

in: service_session_id

in: code_module_id

in: code_configuration

getModuleState

returns the state of the specified service code module (loaded, configured, running).

in: service_session_id

in: code_module_id

out: module_state

changeModuleState

change module state to run, stopped, suspend, terminate

in: service_session_id

in: code_module_id

in: module_state

3.3.3.6 ASP Monitoring

ASP monitoring will be performed by using the monitoring facility from network management architecture (please refer to chapter 2 for details). Any information on the changing status resulting from using the ASP system is given according to the specified use cases (ASP-A: Caching Policy, ASP-B: Code Request, ASP-C: Code fetching/installing, ASP-D: Obstruction clearance, ASP-E: Resource Monitoring)..

3.3.3.6.1 Interface functions

For the monitoring of ASP-related information, the monitoring service of the FAIN Network Management System will be used. A more detailed specification of the monitoring service interface is given in chapter 2.

3.3.4 Execution Environments and ASP

3.3.4.1 Dynamic Provisioning with the DPE-based approach

In the DPE-based approach there is no in-band signalling, i.e. all active code is deployed and managed through a separate control plane.

The main task of the provisioned parts is to manage resources in a specific way to provide a service to applications rather than implement a protocol and handle data packets directly. Thus one would talk about a dynamic service provisioning although it is true that a service also implements a protocol for interacting with the service.

The provisioning of new components on a node when an application needs a specific service isn't triggered by data packets but on a higher plane: the application sends a request for a specific service to the node or group of nodes along a path and the service will be deployed on demand. This can be seen as an explicit provisioning in comparison to the implicit provisioning triggered by data packets.

In the DPE-based approach an object-oriented view is supported where objects interact at interfaces and communicate through bindings. This allows putting the interaction of the application with active nodes inside the object bindings and thus providing an implicit and transparent way for dynamic service provisioning.

We note that in the DPE-based approach there is no in-band signalling, i.e. all active code is deployed and managed through a separate control plane.

The procedure to find the relevant active nodes where service components are needed to fulfil some application's requirements can be quite complicated. It depends on the needed service, the knowledge available to the application, the complexity of the network, and so on. However there has to be a special component on every active node to manage the installed services and on a higher level, another component managing several nodes to ensure the integrity of installed service components. For the sake of scalability a hierarchical model should be adopted.

For clarity we consider one possible scenario with three entities involved: a network provider offering some active services on some of its nodes (e.g. a flow-based reservation service), an application service provider offering a service to applications (e.g. a video-on-demand service), and a client using the application service and transparently interacting with active services to get some desired QoS.

The scenario could work like this: an object of the client's application needs to use an application server's interface. When it requests the possibly personalized interface specifying a desired QoS from the server, the server will create a reference for that interface. Inside the reference, the server can encode information necessary to interact with service components on intermediate active nodes. The reference is passed to the client and decoded by the client's DPE. The DPE will transparently extract the necessary information from the reference and start the interaction with the active service components as previously specified by the application server. Following this, the application's client-server interaction can take place and the QoS can be established by active service components along the path.

3.3.4.2 Dynamic Provisioning with Mobile Agents

Mobile agents are an additional approach for execution of services and related objects in Active Networks. Most tasks and functions of mobile agents in active networks are located in the management and control plane.

Their function is two-fold: (1) well-controlled execution of commands as issued from (i.e. programmed in) service objects and (2) automation and support of control and management tasks. In detail the following tasks can be distinguished:

- Detection/Analysis of problems with different installations of service components (inconsistency in their versions, malconfigured installations).

- Overview of installation/configuration of service components
- Delivery and deployment of new versions of service components (and increments of them)

Typically, the execution environment for mobile agents (MA EE) and as will be used for FAIN is IKV's Grasshopper, which is implemented in Java and based upon a Java virtual machine (Jvm). The MA EE is or will be running in conjunction with the DPE in order to share the Jvm and resource control components. The DPE is described in detail above.

For service provisioning/deployment the installation of service components will be performed in a policy-controlled way from internal or external repositories. For description of service components the OMG Components deployment submissions will be studied [OMG-Comp99]. The policies are defined for resource usage, allowed behaviour of a component and the overall system. A Security Manager is consulted for all security critical activities. The level of granularity of security checks influences directly the overall system and service performance.

A service component can itself be mobile (e.g. an agent) or can be transferred to an active node (AN) by other third entities. An Active Component Manager (ACM) which is part of the MA EE allows AN entities (e.g. users, administrators etc) to install service components on the node and make use of it or possibly make it available to other third party entities via a policy controlled way.

In the MA EE will be in addition to the Active Component Manager a Service Component Repository, Security Manager (Policy Base) and an Audit Manager.

3.3.4.3 High Performance EE

As described previously, there exist two approaches – in-band and out-of-band - to load code into an execution environment. This section discusses the two mechanisms from a high performance viewpoint. In the high performance execution environment we focus on the maximisation of packet throughput. Given the minimum packet size, the link bandwidth, and the clock frequency of the network processor, we find the number of processor cycles available to perform computations on packets at line speed, which is relatively low with currently available technology. Therefore it is important to make efficient use of the limited number of processor commands that can be executed.

In the in-band (capsule) approach code and data are in the same packet. When a packet (capsule) arrives at the node, it is classified and sent to appropriate execution environment. The code is then extracted and loaded into the program memory of the network (co-)processor. For a network processor, such as the Intel IXP1200, this can be a time-consuming task [101]. In order to circumvent this drawback, special purpose hardware [8] has been developed.

Many services suitable for active networking - such as active reliable multicast, virtual private networks, media scaling, etc. - do not require code that is different for each packet. Therefore it is more efficient to once load the code into the execution environment(s) before the packets the code is to be applied on arrive. As a consequence the out-of-band approach avoids bandwidth and computing overhead the in-band approach is suffering from.

In the FAIN enterprise model, service provider and consumer are two separate entities. The service provider deploys services in the network. A service consists of code that is to be executed in execution environments. This model is best supported by an out-of-band approach. The service provider first installs the service (code) in the appropriate execution environments. In a second step the consumer injects packets into the network the service (code) is to be applied on. Using the in-band approach (packets consisting of code and data) service provider and consumer need to work together for the composition of each packet. This approach is reasonable if a single entity plays the role of the service provider **and** customer or if the code is prebound with proper credentials and as such used by other users.

3.4 INFORMATION MODEL

This section will give an overview about how the information model for deployment of service/protocol code as used for ASP mechanisms will look like

Priorities to achieve this are a naming system for services and code modules. We should support different versions and description of dependencies (other services or specific EE type).

3.4.1 Overview on Deployment Descriptions

One effort to create a standard software deployment schema is called the Open Software Description (OSD) format. This effort is a collaboration between Microsoft and Marimba to create a schema for describing software systems for "push" technologies. OSD just allows for the description of multiple coarse-grain variants of a single revision of a software system; dependent software systems may also be specified. The descriptive information includes some identification information and pointers to archives where the code is found. The resulting description is too simplistic to perform any significant software deployment process automation.

The Desktop Management Task Force (DMTF) has created the Management Information Format (MIF). It is a modelling language for describing various computing system elements. DMTF formed a specific working group to create a standard schema in MIF for describing software systems.

3.4.2 FAIN Deployment Description

The ASP defines a syntax for the specification of *service templates* and *service profiles*. The ASP requires this information for the creation, configuration and deployment of each service. The service template contains the necessary common creation information of service instances, whereas the information identified within a service profile is associated with a specific service instance and service user. Both pieces of information serve as input for the ASP.

Since the service template is related to the information required by the ASP to build service instances of a particular service type, this information must be provided by the service designer himself/herself. However, the information is in general, applicable to all service instances.

The figure below depicts the relationship between these two pieces of information and the ASP.

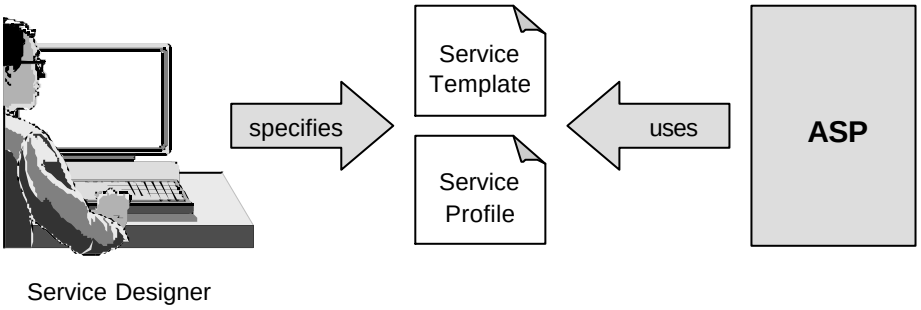

Figure 54 – Input Information for ASP

The following sections provide detailed information about the service template and service profile.

3.4.3 Service Template

A service template specifies the information needed to create a service instance of a particular service type. Among other related information and particularly within the defined scope of FAIN services, it identifies the parts (agents, etc.) comprising a service type. Template descriptions are node independent. Node specific information is maintained in service profiles, which are explained below. At a first glance, a service type is characterised by:

- A *name* which identifies the service type,
- A *code base* identifying the location of the computational information for building the objects composing the service type,
- *service objects/agents*, i.e. computational entities that represent the service type

Name

The name is used to address a particular service type. This name shall be unique within a service provider domain, and it is specified by the service designer. Once specified, such a name is associated with the computational behaviour of a service type. As such, the name generally contains the identification of the components that represent the service, the set of interfaces, and the computational semantic. By having such a name, an abstraction of the details is possible from a user's point of view. Different versions of a service (template) can be expressed either by choosing different names for different versions or by introduction of a version property which allow to differ between various versions of a service template.

Code Base

The purpose of the code base is to identify the locations where the ASP can find the computational specification (e.g. classes, of agents) that represent the service type specified within the service template. A code base shall only be specified if the required computational specification cannot be retrieved by the ASP using local means, i.e. if the required classes are not included by the search path.

Service Agent

All agents comprising a service have to be specified in the service template. The computational specification of each identified service agent must either be retrievable by searching locally or via the code base. If one of the specified service agents cannot be created, the entire runtime representation of the service type will not be built by the ASP.

3.4.4 Service Profile

Whereas the service template covers information that is associated with a service type, the service profile describes a single service instance in particular AN nodes through concrete configuration parameter values and specific information about the initial location of the single service objects/agents.

The following information is covered by each service profile:

Instance Identifier

This entry identifies the service instance uniquely.

Location Information

All agents belonging to a service type are identified in the corresponding service template. At least a subset of these agents has to be deployed by the ASP when a new service instance is created. For those agents that have to be initially deployed by the ASP, location information must be provided within the service profile.

The location information is represented by means of a node or agency name. Additional node requirements or dependencies can be expressed.

Note that those objects/agents, which are specified in the service template and which do *not* have a location information entry within the service profile, will not be created by the ASP when the service instance is created. This case may occur for agents created by service instance components during the service provisioning phase.

Customer Name

This entry represents the name of the customer[31] who has subscribed to the service. This information may be required for authentication purposes.

Customer Identifier

This entry specifies the identifier of the customer who has subscribed to the service. In contrast to the customer name, this entry has to be unique. This information may be required for authentication purposes.

Values of Initialisation Attributes

When creating a new service instance, the single service object/agent may require properties for their initial configuration. The names of these properties are specified for each object/agent within the service. Within the service profile, concrete values can be assigned to these properties.

Textual Description

The creator of the service profile may include a textual description, which provides additional, instance-related information. Note that this description is not interpreted by the ASP.

3.4.5 Node information/characteristics

Each node maintains a registry or database, which describes its properties or characteristics in order to find the right service code (i.e. code base) type for that node. The following properties are suggested:

- Node OS Type and version (e.g. “Linux x.y.z”, “Windows 2000 rev. x.y.z”)
- Software packages (e.g. „JDK 1.3.1“)
- Hardware/Network adaptors (e.g. “ATM Adaptor xzy”)

Please note that the last (Hardware) class of properties is “fixed” in contrast to the other property classes, which allow updates and downloading if required.

3.4.6 Network Information/Characteristics

From a network perspective it must be possible to specify a group or graph of nodes on which service code has to be made available in order to provide the service to customers. It is specified by the Network Manager and implements service provisioning QoS in a best-effort or all-or-none characteristic.

A detailed specification of the information structures explained above maybe expressed using XML and will be embedded in the FAIN overall management information model.

3.5 ASP REQUIREMENTS TO SUBSYSTEMS

[31] A customer may be a subscriber or an end user.

3.5.1 ASP Requirements on the AN Node

The ASP module deals with the provision of active code. The code is delivered to the active node, where it is injected and executed in the appropriate VE. Therefore the active node should provide the facilities for the installation and the execution of the code. In order to support the ASP process, an active node should fulfil a set of requirements. Some of these requirements are expected to be implemented as basic features of the active node.

- Resource Control – Monitoring. The active node should have the ability to allocate computational and network resources to an active service. In addition it should monitor the resources consumed by a service, to ensure that the service is restricted to the predefined resources.
- Code Isolation. The active node should guarantee that there is no interference between different code modules.
- Multiple protocol stacks. An active node should be able to support more than one protocol stack at the same time. Each stack could be associated with a specific user or service of the active node.
- Stack modification. When installing a new protocol, it should be possible to specify the place where the code will be installed. In terms of place we mean the corresponding Virtual Environment, as well as the place of the protocol inside the stack of the active node.
- Protocol/service removal. The ASP can remove protocol/service code, either after an explicit user request, or because a fault has been detected during the execution of the code.

3.5.1.1 Required interfaces from AN Node

The AN Node interface provided to the ASP system should support the installation and removal of active code. Optionally, there could also be operations to suspend and resume the execution of an active code module on-the-fly, without necessarily uninstalling/reinstalling it.

The following interface functions will be needed

installCode (in Code, in Pointer)

This function installs a code module in the active node. The Pointer parameter, which has been obtained by the management system points to the place where the code should be installed (identifies the appropriate VE and the resources which have been reserved by the management system).

removeCode (in CodeId)

This function removes a code module from the code stack of the active node

suspendCode (in CodeId)

This function suspends the execution of an installed module, without uninstalling it from the code stack of the active node.

stopCode (In CodeID)

This function stops the code while it remains installed on the node.

resumeCode (in CodeId)

This function resumes the execution of a suspended code module.

getInstalledModules(out CodeInfoList)

This function gets information about the code modules, which are currently installed in the node. The CodeInfoList contains the identifiers of the installed code modules and other related information, as the status of the code modules (executing, suspended).

3.5.2 ASP Requirements to Security Framework

The security framework must ensure secure service provisioning. To achieve this the security framework should provide the following features:

- Access control, all requests should be properly authenticated, authorized and the security policy should be enforced
- Protect active requests while traversing the network, e.g. provide active code, active packet and possible policies integrity, protect against various network attacks like replay-attacks
- Be able to uniquely identify users request (towards privileged interfaces, messages) while user code is running on the node,
- Be able to uniquely label all objects in the system, including user services installed via ASP,
- Be able to track system state,
- Send terminate requests if the security policies are violated and actually terminate services if particular EE don't respond to terminate request properly,
- Provide audit information, even for user installed services if required and allowed by security policies.

3.5.3 ASP Requirements to Management System

The process of Active Service Provisioning is strongly related to the FAIN management system. The active code that will be deployed by the ASP implements a network protocol or service, which is a target for the management system. The management system is required to perform the necessary checks for access and admission control. The management system will first perform the required checks and then pass the code request to the ASP.

- Access control

  When a request for the installation of code is received, the first thing that should be done is to perform the necessary checks for the authorization of the request.

- Resource Reservation

  In the case that the service/protocol/code that will be installed has specific resource requirements, the management system should take care to reserve these resources on all the nodes that will be used by the service. In case that not enough resources exist, the installation will not be possible.

- Policy Checks

  The Policy Based Network Management system should check all existing policies related to the specific protocol, before giving clearance for the download of code from the ASP system.

- Network Topology Information

Any information regarding the network topology, which may be needed by the ASP system, should be obtained by the management system, to avoid building redundant functionality in both systems.

- Request processing.

  The user of the active network does not directly interact with the ASP system. A request for the installation of a new service should be sent to the management system, where it is processed and then the ASP is subsequently requested to download the code.

- The necessary checks will be performed by the management system before requesting the code download from the ASP, but in the case the dependency resolution, it may decide that an additional protocol is required. In this case, the ASP must request from the management system the authorization and resource allocation for the new protocol.

3.5.3.1 Required Interfaces from the Management System

Typically, it is the FAIN Management System that processes an incoming request for the installation of a new service or code module, reserves the required resources and then asks the ASP system to install the code. In the case that the ASP decides that the installation of an additional code module is required, the Management System interface should be able to receive a request from ASP.

installACReq (in CodeId, in Credentials, in Resources, out Pointer)

This function receives a request from the ASP for the installation of a code module identified by its CodeId. The Credentials of the user who will run this module and the required Resources are also provided as function parameters. The Management System will return a Pointer which points to the place where the code module will have to be installed.

3.6 ASP SCENARIOS

In the following section we will describe a scenario for Active Service Provisioning The scenario itself is much broader than that needed for only the ASP case, but we feel that it is useful to show the entire picture of the code and service lifetime on the node. This can help to get broader understanding of the system behaviour and position of the ASP in the system. The entire scenario is represented in the figure Figure 55. The scenario tries to cover both cases, and in-band and an out-of-band code scenario. On the drawing with numbers in red circles the in-band scenario is presented and with numbers in yellow boxes, in addition to steps in circles, the simple out-of-band approach when the code is referenced in the packet. Specific steps of the management-based approach are shown with numbers in diamonds. The figure also shows the Security block and RCF block which are repeated three times to help to keep the picture more readable. Interactions between these two subsystems are not shown in the scenario. The dashed box in the middle of the diagram is the VE/EE area where many crossings of the arrows over the box boundary can be understood as NodeOS API calls. Light, dashed lines in the lower left corner indicate which parts of the system are in the network. Both approaches, in-band and out-of-band, are valid for ASP scenario and are covered appropriately.

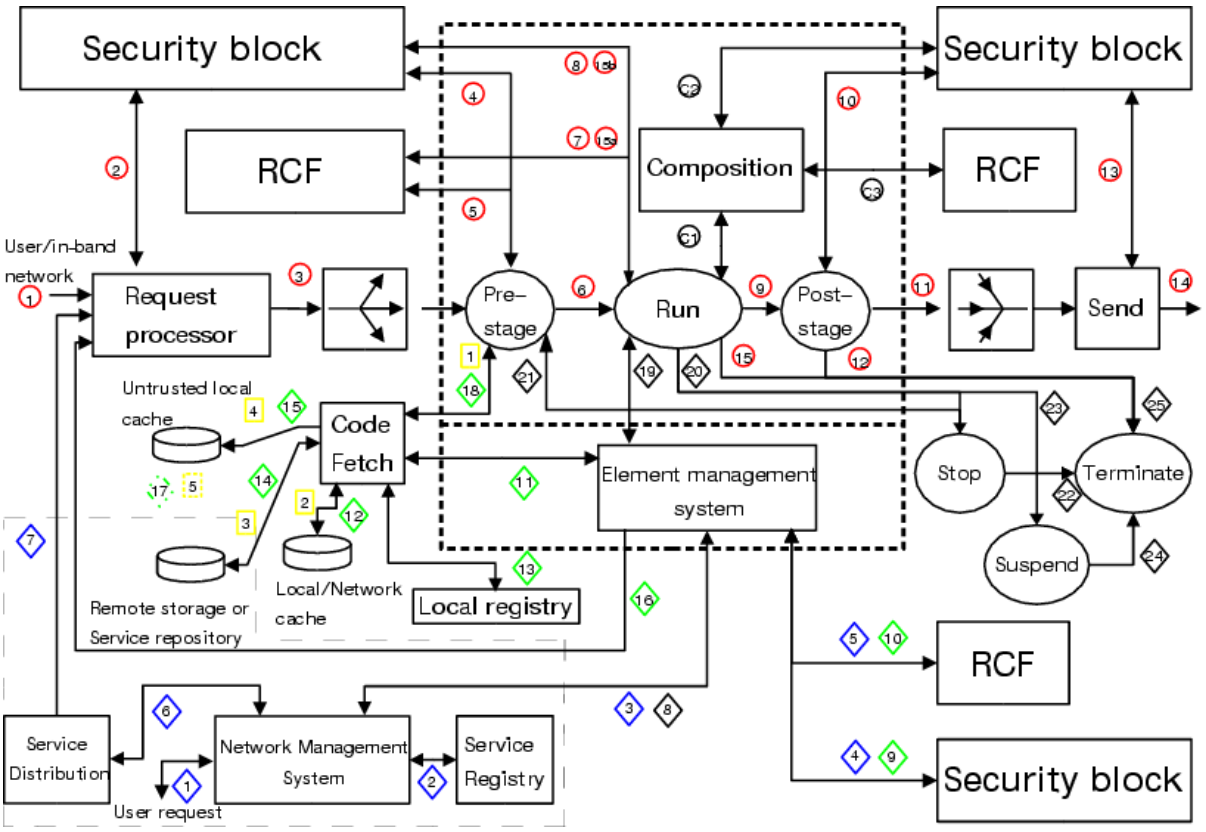

Figure 55 – ASP Scenario

3.6.1 In-band versus out-of-band

The first valid question is what should be considered as the in-band and what as the out-of-band approach.

In-band means that the code is carried in the packet itself and these packets are sometimes called capsules. The usual assumption for this approach is that the capsules are executed on all suitable ANNs in the path. Code in the packet is executed in the suitable node environment and after preparing suitable packet(s) to be send to the network interface(es) packet terminates. Usually the time of the code execution is small and even limited in some approaches. Because of the packet(code) small size the usually in-band approach has support functions in its native environment which should be considered privileged. In this case the code is highly dynamic.

Out-of-band means that the code is not carried in the packets. Code is brought to the node from the network. To carry out this task many solutions has been proposed, from referenced code, network management approach, active agents, DPE ... In this study we will consider only the referenced code and management approach.

In referenced code approach, code to be executed is referenced in the active packet. The reference could be to the previous node, to a trusted repository or elsewhere. The code here can be highly dynamic, as in the case of ANTS for example, when the code is ``walking" through the nodes and is installed from the previous hop network cache if it is not available on the node already [95]. Though the code behaviour is similar to the in-band approach we consider this solution as out-of-band. On the other hand, code can be referenced in the packet but only allowed to run for the time of the certain flow for example [96], or the referenced code can be installed on the node and running for a certain period as defined by the network conditions, the type of the service code is offering, the management actions etc [97]. Code dynamic is various from high to low. In first two cases we know when the code will terminate and these cases can be seen as a very natural way to get the code in the network. In the third case, we don't know how long the code will run on the node, but we have to control its operation; this has to be done through the node and network management mechanisms. The same is valid for code carrying dependency information, since we can get dependency loops. The same is valid for all the other problems mentioned in the “Obstruction clearance” section of this chapter. Possible scenarios to solve problems mentioned in that section are not shown in the scenario but should be solved on network level, probably by Network Management Subsystem. We will call first two approaches simple and these two don’t need in general to interact with management system. The section “Referenced code” given previously will cover these two approaches, while the section “Service initiated by the management system” will cover the third case.

Management approaches differ in the previous solutions in the way the service is triggered and installed on the node; management facilities can request code installation and will lookup upon the code time of running, environment, termination etc. Management approaches can also share many similarities with the referenced code, as we will show later.

3.6.2 In-band approach

First the in-band approach will be studied. Code is carried in the packet itself. Active packet can be generated in the user's machine,[32] or by any other code running on the ANN.

Requests come from the network through network interfaces, so:

1. Requests have to be recognized. This is the first job of the Request Processor, a logical functional block, which job is to recognize and process a request and available information about it. The output from this block must be a validated request bound to a certain security context, which can be sent to appropriate VE/EE.
2. Upon the request many security checks have to be made;
    a. Is there the right packet on the right interface policy check, check against network policies,
    b. Authentication of the packets origin,
    c. Check hop-by-hop active packet integrity,
    d. verification check, check against system policies for general access,
    e. Assign the request on the basis of authentication to the proper Security Context (SC), and Security ID (SID),[33]
    f. Check the VE/EE policies for VE/EE governed access policies,
3. Demultiplexing of the packets to the appropriate VE/EE,
4. Pre-stage is the state of the system when we can do various actions which are VE/EE specific, for example actions that cannot be done at NodeOS level because information about them is not available at previous stages or has no real meaning to the NodeOS itself.

    Requests might have specific policies which can be bounded to active packets and have to be dynamically inserted and checked for conflict with already existing policies.

[32] Machine or device, anything that can run active stack and generate valid active code.

[33] SC should contain only the least privileges for the code, class of the service or type of the VE/EE; any other request to privileged operations, API calls must be authorized when requested.

5. Requests for resources can be checked at this stage. Request can be implicit in the example of the SNAP code[34] or explicitly defined in the request.
6. The code is contained in the packet and is extracted and prepared for execution.

   An environment suitable for code execution is set-up and the code will be run.
7. During the code execution code usage of the node resources has to be monitored and enforced against node, VE, EE and code policy.
8. Any request for privileged access has to be authorized and if granted, it has to be revoked when it is not needed any more.
9. When the code has finished execution it will build active packet(s), which will be send to one/some ANN(s) through the node's interface(s).
10. The code has arrived into the Post-stage state where a system does necessary checks which will enable AC to send itself to one/some network destinations.

    Sending the AC to one/some ANNs through the node's interfaces has to be checked against node, VE, EE and code policy.
11. An active packet is sent to the network interface(s) and multiplexed with other packets.
12. The natural state of the code is termination. The execution environment is responsible for all clean-up of the environment and all used resources must be reclaimed.
13. Packet hop-by-hop integrity information is added to the active packet.
14. Packet is sent to the wire.
15. Up until now all the steps were a valid life cycle of the active packet in the node. This step to the stage of the Termination is added because we must be able to stop the code running due to:
    a. resource issues. AC can use too many resources or it is running too long and
    b. security issues. AC can try to violate security policy.

    Actually only the security block can send a request for termination based on data or triggers from RCF block. All terminated AC resources has to be reclaimed and the environment cleared by the execution environment.

    We could use the management block for the same task but due to scheduling issues and the properties of the in-band approach we are assuming that this is the right solution. The management block can be used in the out-of-band scenario, where code has a longer life cycle to terminate the service, see step 18, or change service state, see steps from 19 to 25 all from the management scenario.

3.6.3 Out-of-band approach

The out-of-band approach as explained in section 3.6.1 will be studied in two stages; first when the AC carries the reference to the code and then as a management initiated task. In addition a management block can introduce in conjunction with the RCF framework implicit requests, which will be explained in section 3.6.3.3 .

[34] The length of the packet is proportional to the estimated max resource usage

3.6.3.1 Referenced code

In this case code is referenced in the packet an as such is not available at the step 1. We can have two options; we can load the code immediately or process the request to the stage when we can see if the code is already available in the system (ANN). We have decided for the latter case because the code might already be in the local cache or in the trusted network cache in the previous network hop. In the cases when the assumption is not true we have to put the code for some amount of time in the untrusted local cache and if needed, check its security properties and only then make code available for execution, i.e.put it in the node local cache. There are few cases when the code can be assumed not to be trusted: when the code is loaded from the user or any other network repository or when the code is crossing a network domain boundary. How to deal with such code is shown in steps from 1 to 5.

Steps from the section 3.6.2 , in-band approach, 1 to 5 are repeated. Then we start from step 1;[35]

1. In the case of the referenced code, requests should be made to the Code Fetch block. Code fetch is a functional block which receives requests for code and looks for it in appropriate places. Code to be fetched, if not available in the local or trusted network cache, is stored in the untrusted local cache.

   After performing the necessary steps from 2 to 5 the code fetch block will return code to the requesting EE.

2. Lookup is first made in the local and then in the trusted network cache.[36] There is no need to do all security checks on this code, though revocation problems might still exist. Caches can be built per user, or if the code can be uniquely named one cache can be used..

3. In already mentioned cases, first, when the code has to be loaded from the user local machine (or referenced predefined repository), and second, on the network boundary, code has to be loaded from remote repository. Loading can be protected by security mechanisms already existing in AN or by other mechanisms like IPsec, SSL ... From where the code can be loaded can be governed by a node policy.

4. Code is stored in a local untrusted cache.

5. The code has to be delivered to the EE and stored in the local cache. Before that some security checks have to be made, e.g. verification checks to verify the code digital signature. Step 5 is not shown in the scenario diagram, it is only marked with dashed box.

   From here on we can proceed with steps 6 to 9 from the in-band scenario. Code can now actually build an AC with referenced code and send it to one/some/allowed interface(s) through the steps from 10 to 11 and 13 to 14 from in-band approach.

   Code can terminate normally through the circled stage 9 and then proceed to the stage Terminate in step 12. All used resources should be reclaimed at this point and the possible states of the code cleared.

   If the code is not behaving properly as explained in item 15, in-band approach, the security block can request code termination. Also at this stage the system has to assure that all the used resources are reclaimed and the states of the code are cleared.

[35] Remember, these steps are shown with a number in a box

[36] Trusted ANN from previous hop.

3.6.3.2 Service initiated by the management system

The management system should be used when it is not known exactly when the service will terminate or the reasons to start the service are not as explicitly given.. On the other hand service provisioning can be too complex to be handled by a simple referenced approach alone. In this case we have to handle various problems described in "Obstruction clearance" use case like dependencies, versioning, setup interference etc. Management actions can be explicit, e.g. when the user sends a request over the network management system to install a new service in the ANN(s) or implicit, triggered by the RCF block or, if based on more complex characteristics estimated from the RCF block data, by a management based trigger and handled in the similar manner as the explicit request. We have divided the scenario in two cases: in the blue diamonds case when the service is initiated by Network Management System is shown and in the green diamonds case when the service is initiated by Element Management System. With black diamonds are marked steps that can change service state and are mainly common to both cases.

An out-of-band AC life cycle can be summarized in the following steps:

1. User request. A user in the system makes the request to provide an active service in the network. The user must be properly authenticated and authorized for such a request.
2. The Network Management system makes a lookup in Service Registry to find suitable code module(s) to fulfil the request.
3. In this step Network Management System sends a request to the Element Management System to reserve needed resources on the ANN(s).
4. The request is checked by the security framework.
5. Availability of the resources are checked in Element Management System and Resource Control Framework.
6. The Network Management System sends a request for distribution to the Service Distribution block. It depends on service distribution methods how the code will be delivered to the selected ANN(s).
7. In this scenario only the method via referenced code is shown. The Service Distribution block will build suitable request(s) with the reference to all needed data to the Request processor. Steps from 2 to 3 are repeated as given in the in-band scenario are repeated and then needed steps from 1 to 5 from referenced code scenario (yellow boxes) are repeated. Then the code is prepared for installation in the Pre-stage. Steps 4 and 5 from the in-band scenario in this case are optional, since resources are already reserved on the node and hence there is no real need for policy installation. The service is then run and steps 7 and 8 (in-band) are valid for this case. Normal service termination is through Post-stage, when used resources are reclaimed and service goes in step 12 (in-band) to state terminate. Step 15 (in-band) is valid if the code is not behaving properly and service will be forced in state terminate as is described in this step.
8. The Network Management System can control the service state through step 8 and set the service to the state Stop, Suspend, Terminate or Run.
9. From this step on we will describe scenario when the service provisioning is initiated by the Element Management System. The trigger in the Element Management System can trigger and actually request service provisioning. These steps are marked with green diamonds. For this approach, the Element Management System should know where to find the triggered service and optionally how many resources the service will use. Typical classes of such service are services for congestions prevention, automated services for Reliable Multicast NACK suppression and cache provisioning etc. In step 9 the security framework is queried for the validity of the request, for example checking the time and node state.
10. The Resource Control Framework is checked for available resources and requests for resource reservation can be made.

11. A request for code modules is sent to Code Fetch block with the location where the service can be found. The following steps are duplicated from referenced code approach (1-5, yellow boxes) because they are slightly different.
12. First a lookup for a code module is done in the local cache. If this step is not successful we proceed to step
13. The Code Fetch block will look in the Local registry if there is more information about the code modules location. Code can be cached in the neighbour node.
14. If the code is not available, for example if the cache time of the code module has expired, code has to be fetched from the Service Repository.
15. The Code module is put in the local untrusted cache.
16. At this time code is already on the node. There can be many ways to actually install and start the code. In this scenario we will show only one simple way when the Element Management System builds a request with a referenced code with reference to local untrusted cache and sends it to the loopback interface of the node. The request is then intercepted by the request processor and all necessary checks in step 2 (in-band) will be done because this cache is considered untrusted.
17. This step is only marked with dashed diamond. During this step the code module after being checked is transferred from untrusted local cache to the local cache and the code modules information in the Local Registry is updated.
18. In this step, the code module is transferred to Pre-stage and subsequently follows the same scenario as described in part of the step 7 of this NMS scenario. Steps 4 and 5 from in-band scenario in this case are optional, resources can be already reserved on the node and there is no real need for policy installation. The service is then run and steps 7 and 8 (in-band) are valid for this case. Normal service termination is through Post-stage, when used resources are reclaimed and the service goes in step 12 (in-band) to state terminate. Step 15 (in-band) is valid if the code is not behaving properly and service will be forced in state terminate.. Actually at this case this can be done by security framework or Element Management System if it has more information on proper service behaviour.
19. Through this step the Element Manager System can control service state and set the service to the state Stop, Suspend, Terminate or Run.
20. Steps 20 to 25 are shown in the scenario to cover other possible service states. The request for state changing is coming from the NMS or EMS, marked as step 8 and 19 in this scenario. In step 20 the request is sent to stop the service. This state is similar to the state terminate but the code is not removed from the system.
21. The stopped service can get to the run state through the step 21, through state Pre-Stage.
22. In this step the stopped service has to change to state terminate if requested. The code module used to provide the service is removed.
23. Step 23 changes the state of the service to suspended. In this case service won't run any more but no resources are reclaimed at this time. Change to state run doesn't require any additional checks from the ANN subsystems. This change should be indicated by back arrow, but is omitted in the figure. The security framework is responsible to watch the possible policy change and change the service state to the stopped state if the security policies regarding the service provided are changed. Status in management system about service status also has to be updated.
24. A suspended service can change to the state terminate if the management system requests this action. At this time all the resources have to be reclaimed and installed service code modules removed from the system.
25. This step is added for the sake of the completeness. The service can be set to the state terminate while running through Post-stage. All used resources have to be reclaimed by the system.

3.6.4 Composition

Composition of services will be covered if necessary in the next stages of the ASP development so at this stage only composition block and some connections to the Security block and RCF are shown on the picture but not explained in the scenario.

3.7 ASSOCIATION BETWEEN ASP AND FAIN ENTERPRISE MODEL

In this section a mapping of ASP to the FAIN Enterprise model (Figure 6) is elaborated. It analyses mainly the roles and reference points to which the ASP and its activity are mapped.

3.7.1 Mapping ASP to FAIN Enterprise model Actors

The functionality of the ASP can be mapped to the Network Infrastructure Provider (NIP) of the FAIN Enterprise model. The NIP is the entity that offers a basic platform to the Active Network Service Providers (NSPs), who can then build their own services on top of it. We consider the ability to dynamically provide active code for the implementation of new services a basic facility of the active network, thus we believe that the ASP can be mapped to the NIP. Furthermore, the ASP is a generic mechanism for the injection of service code, so mapping it to the NIP allows us to have a single mechanism for code provision. Otherwise, it would be necessary to have different ASP systems for each NSP. The other actors who want to install their code in the network, the SP and the NSP, have to implement some higher-level mechanisms of their own and use the ASP facility of the NIP to do the actual injection of the active code in the system.

3.7.2 Mapping FAIN enterprise model RPs to ASP functions

The reference points of the Enterprise Model, which are directly related to the ASP, are RP3 and RP7. Since the interdomain RP7 has not been covered in detail in the FAIN Enterprise Model, the ASP functions can be mapped only to RP3. We should also mention that the FAIN Management System is also owned by the NIP, so all interactions between the ASP and the PBNM system are considered as internal to the NIP and hence don't map to any of the reference points.

3.7.2.1 RP3

- The functionality of the ASP which can be mapped to this reference point is the following: A new service can be registered and its corresponding implementation can be stored in the code repository, from where it can be retrieved and installed by the ASP system.
- Additional requirements for the service can be given, like dependencies on other services or code modules, requirements for specific types of Execution Environments.
- Information about services, which exist in the system can be retrieved.
- The actual injection of the code is done on behalf of the NSP or the SP who wants to install their own service in the active nodes.

We should also mention that the actual request for the installation of the active code is sent from the user to the management system, so there is not a direct interaction of the user with the ASP system in this reference point.

3.8 FURTHER ISSUES

Some issues for the implementation of the ASP system, as well as for consideration in the next phase of the project are presented below.

- ASP platform

In the scope of FAIN a variety of Execution Environments will be developed (DPE-based, MA-based, High Performance). The active code for each different EE will have its own format, however the distribution of the code can be independent. Our initial goal is to implement a single ASP system for all the types of EEs inside FAIN. Mobile Agents provide a flexible solution for the provision of services and they will be used in the initial implementation of the ASP system. As the project evolves and the different Execution Environments are developed, if new issues arise which demand the existence of other EE-specific mechanisms for the provision of services, we will proceed with the implementation of EE-specific ASP components.

- Caching mechanisms

  The initial implementation of ASP will contain a simple caching scheme, where each active node will cache locally recently used code and upon a request for code installation it will first lookup in the local cache and if the code does not exist then it will lookup in the central code repository.

  In order to improve performance, more advanced caching schemes can be used in future versions of ASP. For example code caching can be distributed in the network, with each active node having knowledge of the contents stored in the caches of its neighbouring nodes.

- Code Dependency and Versioning check

  The checks made to resolve the dependency of an active service on additional code modules, as well as requirements for specific versions of protocols or services can become very complex. In the initial implementation and testing of the ASP system, there will not be an immediate need for very sophisticated dependency and versioning mechanisms. Dependencies and version-specific requirements will be contained in the service requirements entry in the service registry.

- Registry Update

  The mechanisms for the update of the code registry are left for the implementation. The registry could be updated either synchronously, after checking the contents of the code stacks and the local caches of the different active nodes, or asynchronously, when a new service or protocol is installed on an active node, or a service is removed from the code stack or local cache of a node.

4 CONCLUSIONS

This document has provided an initial outline of the case studies that are to be used to evaluate the overall FAIN approach and associated architecture. The work documented here has focused in particular on the development of case studies associated with policy based network management and the dynamic provisioning of services in an active networking domain (as initially presented in the original technical annex). The case studies themselves have been based around the enterprise model developed in WP2. That is, the same key actors and roles have been re-used here and the scenarios showing their more detailed interaction presented.

Work is currently on-going in the development of prototypes that realise these case studies. Initial successes have already been developed, e.g. prototyping of the VPN scenario, showing that the components as specified here and their interaction scenarios, are both meaningful and realisable in an active networking environment. This work is of course on-going and as such it is likely that the prototypes will undergo further refinements and enhancements as the work on FAIN progresses. As such, it is expected that the prototypes and the scenarios that they support will evolve into a more complete final FAIN demonstration. This will both highlight the success of the general overall approach, i.e. that the FAIN architecture is sufficiently well defined (through generic components and interfaces between the different actors) so as to support a variety of different services and management capabilities in a dynamic (active) manner, as well as to show the overall benefits of an active networking approach for service provisioning and network management.

5 REFERENCES

[1] N. Damianou, N. Dulay, E. Lupu, M. Sloman, 'The Ponder Policy Specification Language', Policy Workshop 2001, HP Labs, Bristol, January 2001

[2] Telecom Operations Map - http://www.tmforum.org/publications/telops

[3] "High Availability Design For Embedded Systems", Wind River Systems http://www.wrs.com/products/html/cirrus-wp.html

[4] R. Yavatkar, D. Pendarakis, R. Guerin, "A Framework for Policy-based Admission Control" – RFC 2753, January 2000

[5] J. Bacon, K. Moody, J. Bates, R. Hayton, Ch. Ma, A. McNeil, O. Seidel, M. Spiteri, "Generic Support for Distributed Applications", IEEE Computer, March 2000.

[6] M. Stevens, W. Weiss, H. Mahon, B. Moore, J. Srassner, G. Waters, A. Weterinen, J. Wheeler, "Policy Framework", IETF draft, September 1999.

[7] S. Denazis, 'AN Reference Architecture', R23 Audit document, Dec 2000

[8] E. Lupu, M. Sloman, "Conflict Analysis for Management Policies", Proceedings of the 5th International Symposium on Integrated Network Management IM'97, San Diego, Chapman & Hall, May 1997

[9] M. Sloman, E. Lupu, 'Policy Specification for Programmable Networks', IWAN '99, Berlin, June 1999

[10] E. Lupu, "A Role-Based Framework for Distributed System Management", Departement of Computing, Imperial College, July 1998.

[11] K. Ho Chan, D. Durham, S. Gai, K. McCloghrie, F. Reichmeyer, J. Seligson, A. Smith, R. Yavatkar, "COPS Usage for Policy Provisioning", Internet Draft, October 2000.

[12] "Interoperable Naming Service Specification", Object Management Group, November 2000

[13] K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita,A. Smith, F. Reichmeyer, "Structure of Policy Provisioning Information (SPPI)", Internet Draft, January 2001.

[14] M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, F. Reichmeyer, "Framework Policy Information Base", Internet Draft, November 2000

[15] M. Blaze, J. Feigenbaum, J. Lacy, 'Decentralised Trust Management', IEEE Symposium on Security and Privacy, Oakland CA, May 1996

[16] IETF "The Java LDAP Application Program Interface", draft-ietf-ldapext-ldap-java-api-13.txt, Feb 2001.

[17] IETF "Policy Core Information Model -- Version 1 Specification", RFC3060, Feb 2001.

[18] CORVAL2 Project, see http://www.opengroup.org/projects/corval2/

[19] I. Schieferdecker: An Approach for Performance Tests of CORBA based Systems. - Workshop on Testing Non-Functional Software Requirements at ConQuest'99, Nuremberg (Germany), Sept. 1999.

[20] I. Schieferdecker, M. Li, A. Rennoch: Incremental Testing at System Reference Points. - The IFIP 13th Intern. Conf. on Testing of Communicating Systems, Ottawa (Canada), Aug. 29 - Sept. 1, 2000.

[21] I. Schieferdecker, M. Li: Functional Tests for Component Based Distributed Systems. - First Intern. Conf. on Software Testing (1st ICSTEST 2000), Bonn (Germany), April 5-7, 2000.

[22] Th. Walter, I. Schieferdecker, J. Grabowski, H. König: Test Architectures for Distributed Systems - State of the Art and Beyond (Invited Paper).- IFIP 11th International Workshop on Testing of Communicating Systems (IWTCS'98), Tomsk (Russia), Sept. 1998.

[23] Information technology – Open Systems Interconnection – Conformance Testing Methodology and Framework – Part 3: The Tree and Tabular Combined Notation (TTCN), ISO/IEC 9646-3 1997 (E).

[24] Lukas et. al., FAIN Workpackage 3 internal report R23.1 " Node Architecture Project Review", Jan. 2001.

[25] BANG project deliverable "Deliverable 7: Requirements and Specification of the Open Programmable IP Router API, V2", March 31st, 2000.

[26] Y. Bernet, et. al., "Interoperation of RSVP/Intserv and Diffserv networks", Internet Draft < draft-ietf-issll-Diffserv-rsvp-02.txt>, June 1999

[27] Diana Rawlins et.al., "RSVP Policy Control Criteria PIB", <draft-rawlins-rsvppcc-pib-00.txt>, November 16, 2000.

[28] M. Fine et.al., "Differentiated Services Quality of Service Policy Information Base", <draft-ietf-diffserv-pib-02.txt>, Nov. 2000.

[29] D. Gabrijelcic, F. Mocilar, A. Savanovic, 'Security Framework for Active Networks and FAIN Security Architecture', Dec 2000

[30] IETF "PCIM extensions", draft-ietf-policy-pcim-ext-00.txt, February 2001

[31] IP VPN Information Model: draft-iyer-ipvpn-infomodel-00.txt

[32] Common Information Model (CIM) Core Model. Version 2.4 http://www.dmtf.org/spec/release/Whitepapers/DSP0111.doc

[33] "XML Schema Part 0: Primer" W3C Proposed Recommendation, 30 March http://www.w3.org/TR/xmlschema-0/

[34] DMTF, "XML As a Representation for Management Information", Sept 1998 http://www.dmtf.org/spec/xmlw.html

[35] DMTF, "CIM Core Policy Model", May 12, 2000. http://www.dmtf.org/spec/release/Whitepapers/DSP0108.doc

[36] "Extensible Markup Language (XML) 1.0 (Second Edition)" W3C Proposed Recommendation, 6 Ocotober 2000. http://www.w3.org/TR/2000/REC-xml-20001006

[37] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus. "Requirements for Traffic Engineering Over MPLS" – RFC 2702

[38] Internet Draft, "Policy QoS Information Model" <draft-ietf-policy-qos-info-model-03.txt>. April 2001.

[39] SOAPRPC http://www.soaprpc.com

[40] XML-RPC http://www.xml-rpc.com

[41] ILOG Rules – http://www.ilog.com/products/rules

[42] Grasshopper - www.grasshopper.de

[43] The Mobile Agent System Interoperability Facility, 1997. ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf.gz

[44] The FIPA home page, www.fipa.org

[45] World Wide Web Consortium – http://www.w3.org

[46] Resource Description Framework – http://www.w3.org/RDF

[47] XML-Schema – http://www.w3.org/XML/Schema

[48] Document Object Model (DOM) – http://www.w3.org/DOM

[49] SAX 2.0: The simple API for XML - http://www.megginson.com/SAX/

[50] XML Spy – http://www.xmlspy.com

[51] XML Software - http://www.xmlsoftware.com/

[52] XML Software: XML parsers - http://www.xmlsoftware.com/parsers/

[53] Java API for XML Processing - http://java.sun.com/xml/download.html

[54] Java XML Parser: Crimson - http://xml.apache.org/crimson/

[55] Xalan - http://xml.apache.org/xalan/index.html

[56] Java Naming and Directory Interface - http://java.sun.com/j2se/1.3/docs/guide/jndi/

[57] Policy Standards and IETF Terminology - http://www.stardust.com/qos/whitepapers/IPHighway_vol2paper/polstandardsIETFterm_04.htm

[58] Simple Object Access Protocol (SOAP) - http://xml.apache.org/soap/

[59] Apache SOAP - http://xml.apache.org/soap/

[60] SOAP for Java - http://www.alphaworks.ibm.com/tech/soap4j

[61] Apache SOAP Features - http://xml.apache.org/soap/features.html

[62] The Mobile Agent System Interoperability Facility, 1997 – ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf.gz

[63] AdventNet: The Internet Management Infrastructure Company – http://adventnet.com

[64] Java Naming and Directory Interface API - http://java.sun.com/j2se/1.3/docs/guide/jndi/spec/jndi/jndi.5.html

[65] "CORBA Notification Service", OMG TC Document, November 1998

[66] R. Kawamura, R. Stadler, "Active distributed network management for IP networks", *Programmable and Active Networks Seminar at Networking 2000*, Paris, May 17, 2000.

[67] D. Raz, Y. Shavitt, "Active Networks for efficient distributed network management", *IEEE Communications Magazine*, pp. 138-143, March, 2000

[68] M. Feridun, "Distributed management with mobile components", *Programmable and Active Networks Seminar at Networking 2000*, Paris, May 17, 2000.

[69] A. Kulkarni, G.J. Minden, V. Frost, J.B. Evans, "Survivability of Active Networking Services", *Active Networks, First International Working Conference, IWAN'99*, Berlin, June 30 – July 2, 1999, Proceedings.

[70] D. Putzolu, S. Bakshi, "The Phoenix Framework: a practical architecture for programmable networks", *IEEE Communications Magazine*, pp. 160-165, March, 2000.

[71] D. Raz, Y. Shavitt, "New models and algorithms for programmable networks", *Technical Report ITD-99-38382S*, Lucent Technologies, November 1999.

[72] J. Kornblum, D. Raz, Y Shavitt, "The active process interaction with its environment", *The second International Working Conference on Active Networks (IWAN2000)*, Tokyo, Japan, October 16-18, 2000, Proceedings

[73] S. F. Bush, "Active Virtual Network Management Prediction", *Trans-European Research and Education Networking Association (TERENA) 2001*, Antalya, Turkey, May 14-17, 2001

[74] M. Brunner, "A Service Management Toolkit for Active Networks", *IFIP/IEEE International Symposium on Network Operations and Management (NOMS 2000)*, Hawaii, USA, 2000

[75] S. Berson, B. Braden, L. Riciulli, "Introduction to the ABone" www.isi.edu/abone/into.html

[76] M. Blaze, J. Feigenbaum, J. Ioannidis, A. Keromytis, 'The KeyNote Trust-Management System Version 2', RFC 2704, September 1999

[77] "Tivoly Distributed Monitoring Whitepaper", Tivoli Systems.

[78] Boyle, J., Cohen, R., Durham, D., Herzog, S., Raja, R., Sastry, A., "The COPS (Common Open Policy Service) Protocol", IETF RFC 2748, January 2000.

[79] P. Lin, S. Denazis, J. Vicente, M. Suzuki, J.P. Redlich, F. Cuervo, J. Biswas, W. Wang, K. Miki, J. Gutierrez: " Programming Interfaces for IP Networks, A White Paper." P1520/TS/IP-001, June 1999.

[80] J. Biswas, J. Vicente, M. Kounavis, D. Villela, M. Lerner, S. Yoshizawa, S. Denazis: "Proposal for IP L-Interface." P1520/TS/IP013, January, 2000

[81] M. Raguparan, J. Vivente: "L-Interface Specification for IP devices." P1520/TS/IP-014, July, 2000

[82] Peterson, L., 'NodeOS Interface Specification', AN NodeOS Working Group, Jan. 2001

[83] S. Blake et.al., "An Architecture for Differentiated Services", RFC2475, December 1998.

[84] R. Braden (ed.), L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional SpecificationRFC 2205, September 1997.

[85] C. Goh, 'A Generic Approach to Policy Description in System Management', Proceedings of the 8 th IFIP/IEEE International Workshop on Distributes Systems Operations and Management (DSOM) 1997

[86] G. Kotonya and I. Sommerville, 'Requirements Engineering - Processes and Techniques': John Wiley, 1998

[87] P. F. Linington, 'An ODP View of Roles, Policies, and Communities', Policy Workshop 1999, HP- Laboratories, Bristol, November 1999

[88] J. D. Moffett and M. S. Sloman, 'Policy Hierarchies for Distributed Systems Management' IEEE Journal on Selected Areas in Communication, vol. 11, pp. 1404-1414, 1993

[89] R. Wies, 'Using a Classification of Management Policies for Policy Specification and Policy Transformation', Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management '95, Santa Barbara, CA, USA.

[90] A.Carzanga, D.S. Rosenblum, A. Wolf, "Interfaces and Algorithms for a wide-Area Event Notification Service"

[91] M. Brunner, J. Quitteck, "Policy Framework Core Info Model Extensions", IETF Draft, November 2000

[92] D. Marriot, M. Sloman, "Implementation of a Management Agent for Interpreting Obligation Policy", Imperial College *Department of Computing*, April 1996

[93] D. Rawlins, A. KulKarni, K. H. Chan, D. Dult, "Framework of COPS-PR Policy Information Base for Accounting Usage", draft-ietf-rap-acct-fr-pib-00.txt, December 2000

[94] K. Sugai, Y. Sako, T. Aimoto, "GR-2000: a Gigabit Router for a Guaranteed Network", Hitachi Review Vol. 48 (1999), No. 4

[95] Savanovic Arso, Gabrijelcic Dusan, and Mocilar Franci. Initial Security Requirements in FAIN. November 2000, https://face.ee.ucl.ac.uk/bscw/bscw.cgi/d27990/WP3-JSIS-001-I23-Int-Sec_Req_V1.0.pdf. Document in FAIN BSCW server

[96] Savanovic Arso, Gabrijelcic Dusan, and Mocilar Franci. Security framework for Active Networks and FAIN Security Architecture. November 2000, https://face.ee.ucl.ac.uk/bscw/bscw.cgi/d55707/WP3-JSIS-002-I23-Int-sec_framew_archit.pdf. Document in FAIN BSCW server

[97] Michael Hicks and Scott Nettles. Active networking means evolution (or enhanced extensibility required), Proceedings of the Second International Working Conference on Active Networks. oktober 2000, http://flashed.cis.upenn.edu/.

[98] FAIN: Future Active IP Networks. Jan 2000, http://face.ee.ucl.ac.uk/fain/. Technical Annex

[99] Desktop Management Task Force, "Enabling your product for manageability with MIF files," Nov. 1994.

[100] A. van Hoff, H. Partovi, T. Thai. "The Open Software Description Format (OSD)," Microsoft Corp. and Marimba, Inc., 1997. http://www.w3.org/TR/NOTE-OSD.html.

[101] T. Spalink et al., Evaluation Network Processors in IP Forwarding, Technical Report TR-626-00, Princeton University, 2000.

[102] T. Egawa, StreamCode: A Hardware-oriented Capsule-type High-performance Active Network, in Proceedings of Stockholm

[103] FAIN Deliverable D1-Requirements Analysis & Overall Architecture

[104] FAIN Deliverable D2-Initial Active Network and Active Node Architecture

6 APPENDIX A : R25

6.1 INTRODUCTION

This report provides an introduction to the issues involved in network management, specifically in the context of active networks. It summarises current non-active approaches, architectures, tools and protocols for network management. An overview of existing work in applying active networks for network management is given and an initial outline of the key issues that are to be addressed in FAIN in applying active networks for policy based network management are provided. This report thus provides the background information for the WP4 case study on policy based network management, as well as a source for the initial issues that are to be resolved in realising this case study.

6.2 INITIAL SUMMARY OF EXISTING NETWORK MANAGEMENT APPROACHES

6.2.1 Existing Network Management Approaches

In this section we provide an outline of the way in which "traditional" network management has been done. In particular we identify architectures and approaches that have been used to perform network management. Specifically, we consider TMN, TMF/NMF, TINA and the IETF/DMTF policy based network management approaches. We also identify the limitations with these approaches and provide arguments for the adoption of the more dynamic approach that can be achieved via active networks.

We also provide a summary of existing policy based network management approaches, tools and protocols that are currently used for network management together with their advantages and disadvantages.

6.2.1.1 TMN

The Telecommunications Management Network (TMN) supports management activities associated with telecommunication networks which consist of many types of analogue and digital telecommunication equipment and associated support equipment. The basis for TMN is the distributed network management. TMN supports numerous operating systems for telecommunication management. Each operating system has management functions which interact with potentially remote network elements. A TMN may provide management functions and communication to another TMN. In ITU-T M.3010 [45], there are three sub-architectures of TMN.

6.2.1.1.1 TMN Functional Architecture

This architecture describes the appropriate distribution of functionality within the TMN. In this architecture, TMN has six functional blocks including the Operations System Function (OSF), Network Element Function (NEF), Workstation Function (WSF), Mediation Function (MF), Q Adapter Function (QAF), for management.

An example of the WSF is a user interface. The QAF can be considered as a kind of gateway for connecting between TMN and non-TMN entities. The MF acts on information passing between an OSF and NEF (or QAF). This function may include storage, adaptation, filtering and so on. NEF is a functional block which communicates with the TMN for the purpose of monitoring and control. The NEF provides the telecommunications and support functions required by telecommunications networks. OSF is the telecommunication management function block. OSF processes information related to the telecommunication management for the purpose of monitoring and controlling telecommunication function including management functions.

The kinds of communication with each other are also defined as reference points in this architecture, namely the q, f, x reference point (see in [45]). The x reference point is the interface between OSFs of two TMNs or between the OSF of a TMN and the equivalent OSF-like functionality of another network.

Concerning the OSF, the architecture classifies the following layer structure.

- **Business management layer:** This layer is concerned with the network planning and agreements between operators. BML functions have more to do with setting and tracking the overall goals than with implementation details.
- **Service management layer:** This layer provides the customer interface. Some of the main functions of this layer are for service order handling, accounting, fault reporting, maintaining statistical data (e.g. QoS) for provided network services to customers etc. The SML also provides interaction with service providers and between services. The SML does not include the management of physical entities.
- **Network management layer:** This layer manages network (end-to-end network) configuration using information provided by network element management. The functions of this layer are to control and coordinate the network view of all network elements within this scope or domain, to maintain network capability and data about the network, to interact with the service management layer, and so on.
- **Network Element management layer:** This layer manages each network element on an individual or group basis and supports an abstraction of the functions provided by the network management layer. This layer has one or more element OSFs and/or MF. The network element management layer is refered to as the Element Manager.

Besides these layers, the Network Element Function is added as **Network Element Layer** in TMN management layer. These layers constitute a hierarchical structure in a TMN. The upper layer will use lower layer interfaces or functions to manage the layer.

Moreover, examples of interaction between multiple TMNs are considered in [45]. One example is that OSFs in service management layer communicate with each other via an x reference point. Another example is that an OSF in service management layer in a TMN communicates with OSFs in service, network, and element management layer in another TMN.

6.2.1.1.2 TMN Information Architecture

This architecture is based on an object-oriented approach and gives the rationale for the application of OSI system management principles to the TMN principles. The information for network management is defined using GDMO template and ASN.1 in ITU-T standards.

6.2.1.1.3 TMN Physical Architecture

This architecture is a physical image based on functional block in telecommunication management. Figure 56 shows an example of a simplified physical architecture. One function block is defined in several physical devices; e.g. OSF may be implemented in more than one OS. There is a DCN (Data Communication Network) between the MD and NE (or QA). The DCN is the network used for interacting between MD and NE (or QA). There are also DCNs between MD and OS. The DCN also connects with QA and/or NE. Both DCN connect with WS for providing information to user. The reference point between the function block in the functional architecture for TMN is implemented as an interface.

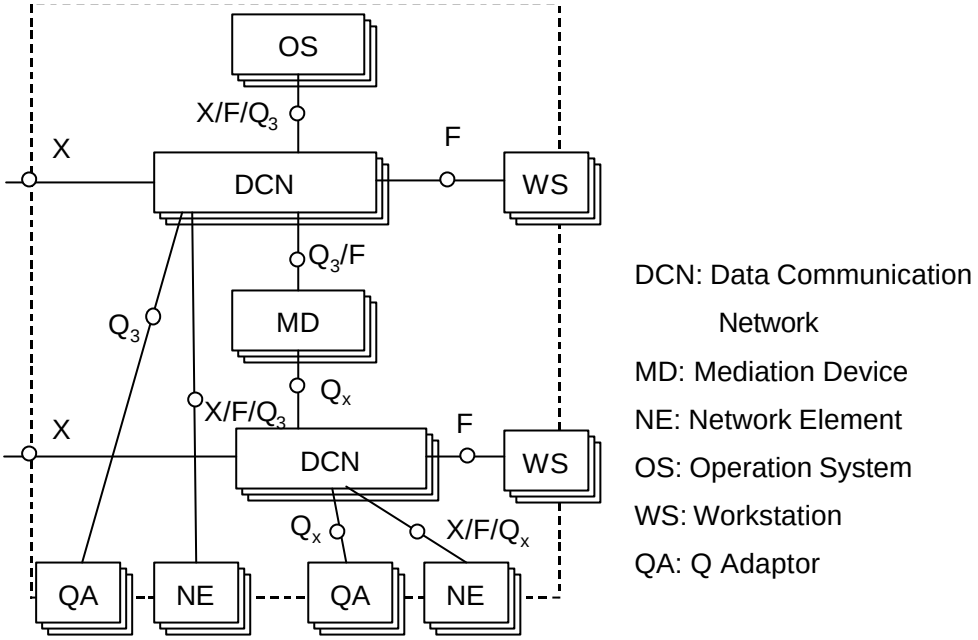

Figure 56 : Physical Architecture of TMN

6.2.1.2 TMF

TeleManagement Forum (TMF - formerly NMF) and its member companies is intended to identify, create, develop, and implement real world solutions that automate and streamline telecom operations. In other words, seek the most effective ways to improve public networks and services management.

The Telemanagement Forum activities are structured in programs with the overall program development being guided by the TM Forum Strategic Plan – a living document that represents a collective vision of likely trends and developments in the communications industry over the next two to three years. The major program areas include the Process Automation Programs and the Technology Integration Programs which are both supported by specific projects called Catalyst Projects.

The TeleManagement Forum's Process Automation Programs are based on the following :

Telecom Operations Map Team

Service Assurance Program

Service Fulfilment Program

- TeleManagement Forum's Telecom Operations Map is the de facto industry view of essential processes required for the fulfilment, quality assurance and billing of telecom services.
- TeleManagement Forum's Service Fulfilment Program's primary purpose is to enable flow-through automation from the customer to the network elements and end-to-end process flow for interfaces required for assurance and billing for services requested by the customer.
- TeleManagement Forum's Service Assurance Program aims to identify common components of Service Level Agreements and takes a process-oriented view of the SLA process.

The TeleManagement Forum's Technology Integration Programs aim to provide development specifications dealing with the integration of information technology (such as between the CMIP/GDMO and CORBA technologies) or with the use of specific technologies such as work flow engines.

The TeleManagement Forum has always taken a pragmatic approach in its conclusions: real world and product solutions are saught, as specifically claimed in the introduction of the Catalyst projects. This entails the use of already existing and quite mature technologies and products.

Unfortunately, the recent developments in the telecommunication arena have shown that customer requirements change rapidly with increasing demands on the infrastructure and services and protocols to be supported there. The process of standardization, development, and deployments of new protocols being a laborious and drawn out one. A representative example of this is the deployment of RSVP and IPv6, which has been widely recognised as useful, however, their network-wide deployment has still not taken place. This is primarily due to the problems that would be incurred when deploying such protocols across the global network. Specifically, their network-wide deployment would require the substitution and updating of software in thousands of routers and potentially millions of hosts. As a result, many innovative applications that would require features of new protocols can not currently be supported. We note here that these new protocols might also be management protocols. Hence, a framework to support such network infrastructure dynamism is required. Unfortunetaly, the TMF work does not lend itself to such dynamism. Instead it relies upon standardised and widely available protocols such as CMIP and SNMP for management, and does not cater for new and innovative application specific management approaches and protocols.

The above considerations lead to the conclusion that dynamical network infrastructures were required to expedite and facilitate the deployment and management of new network protocol software, thereby helping to solve the problems in today's networks. Active networks represent one viable way in which the dynamicity of the network infrastructure can be supported directly.

6.2.1.3 TINA

The TINA Consortium [24] was formed in 1993. Its intention was to define a common architecture upon which next generation telecommunication services could be built. The architecture proposed logically separated high level applications from the physical infrastructure, e.g. the network resources, used by those applications. Central to the architecture was object-oriented distributed architectures such as the Object Management Groups [28] and the support of Distributed Processing Environments (DPE).

The architecture itself was defined through specifically identified components, clear separation points called Reference Points and guidelines on how to structure applications so that they could be easily integratd into the TINA architecture. To help in this regard, specific languages [29] were developed to help overcome some of the limitations of Interface Definition Languages (IDL) for describing the syntactic aspects of distributed systems.

TINA was divided into the following sub-architectures:

- **Computing Architecture** [25] defines modeling concepts and the issues related to the DPE. The DPE resides in heterogeneous pieces of equipment, and, by hiding their distribution, makes them function as a single system for applications. The TINA DPE is based on OMG's CORBA.
- **Service Architecture** [26] defines a set of principles for providing services. It uses a notion of session to offer a coherent view of the various events and relationships taking place during the provision of services. In particular the **access**, **service** and **communication sessions** are defined. The access session represents mechanisms to support access to services (service sessions) that have been subscribed to. The service session includes functionality to execute and control and manage sessions, i.e. it allows control of the communication session. The **communication session** controls communication and network resources required to establish end to end connections.
- **Network Architecture** [27] describes a generic, technology-independent model for setting up connections and managing telecommunication networks. It inherits concepts used in ITU-T and other standards bodies. It extends these concepts to integrate network control and management software for different network technologies.

6.2.1.3.1 TINA Network Management Discussion

The TINA architecture offers certain components that can be applied directly for network management. Unfortunately, the full TINA architecture - the network architecture in particular - is seen by some as being too heavyweight for most applications and services. The architecture is suited for applications that require full end-end control and management of multimedia flows as might be needed in a multimedia conferencing application. For less complex systems, e.g. based around Internet technologies, the architecture offers insufficient insight into how best network management might be achieved from the service or network perspective.

As a result, it has often been the case that subsets of the overall TINA architecture have been implemented. This was typically the case for the Service Architecture and the service session which suggested that services could be constructed through *feature sets*. Many felt that the structuring imposed by feature sets was too restrictive, hence they implemented their services differently, i.e. not through feature sets.

Whilst the full TINA architecture was not completely accepted by the telecommunications and software industry more widely, many of the ideas that were contained within the TINA architecture were seen as useful, and as such are currently being re-used in other areas. An example of this is the adoption of components and reference points from the service architecture, namely: access session related components used to ensure the secure access, personalisation and subsequent usage of (telecommunication) services. Specifically, the OMG Telecommunication Service Access and Subscription [30] work has adopted components that bear a direct counterpart to those existing in the TINA Service Architecture.

Recent works [31,32] have also shown how one such realisation of the TINA architecture [33] have been extended with specific components that allow for direct relations between service and network level management systems to be made. This work was based upon the exchange of trouble tickets between CORBA and TMN implementations via appropriate gateways of service and network level management systems respectively. Of particular relevance to FAIN was how TINA service level components were extended to include service level agreements made on both the services and networks that those services make use of. These agreements were subsequently used to ensure amongst other things, that service quality could be enforced, or when this was not possible, that appropriate discounts were given to the affected users.

6.2.1.4 IETF/DMTF

The use of policies for network management has recently been introduced to the Internet community. However, for the deployment of Policy Based Network Management Systems in the Internet, a standardisation process is required, to ensure the interoperability between equipment from different vendors and PBNM systems from different developers. Both the Internet Engineering Task Force (IETF) and the Distributed Management Task Force (DMTF) are currently working for the definition of standards for Policy Based Network Management. The DMTF is mainly focused on the representation of policies and the specification of a corresponding information model and schema. The IETF is also working in that field, in co-operation with DMTF, while also trying to define a general framework for a PBNM system, as well as a protocol that could be used for implementing a PBNM system.

6.2.1.4.1 DMTF work on Policy Based Network Management

The DMTF has adopted the Directory Enabled Network (DEN) specification. The purpose of DEN is the mapping of network services to a directory. The directory objects can represent networks/subnets, services, devices, applications, users or locations. The relationships of these objects are also defined and managed in the directory. The management data can be accessed through ties to the directory objects. In this way we have a single model for the access to the objects, enabling the interoperability of different management systems. Thus the final goal of DEN is the use of a single directory as a centralised repository, that stores the relationship of users and applications and their relationship to network services, in order to enable management interoperability and information sharing.

The DMTF has defined the Common Information Model (CIM) management schema, which consists of an object-oriented model for the representation of the information that will be stored in the directory of a DEN-enabled network.. The CIM is structured into three layers. The core model captures notions that are common to all areas of management. The common model captures notions that are common to particular management areas, but independent of a particular technology or implementation. The common areas are system, applications and devices. The third layer of the CIM are the extension schemas, which represent technology-specific extensions of the common model.

The CIM is an information model for general management information, that does not have a direct relation with policy based network management. Policy-related work is done in the DMTF by the Service Level Agreement (SLA) Working Group. This group is working in association with the IETF policy workgroup, to extend the CIM syntax and metaschema, to allow the definition and association of policies, rules and expressions. These changes will extend the CIM core model, allowing for a further domain-specific specialisation by the common model, which subclasses the core model.

Except the information model of policies, the DMTF SLA group will also deal with other policy issues, like mechanisms for conflict resolution, the specification of priority and ordering of rules and expressions, authorisation and motivation policies, mechanism for management of policies and rules across multiple policy domains and linkage with events and models to allow detection and response to Policy and Service Level Agreement violations.

So far the SLA working group has not produced any documents that are publicly available.

6.2.1.4.2 IETF work on Policy Based Network Management

The main work of the IETF concerning Policy Based Network Management is done by the IETF Policy Framework (policy) workgroup. The primary goal of this workgroup is to provide a framework for the representation, management, sharing and reusability of policy information in a vendor-independent, interoperable and scalable manner. The next step is the definition of a core information model and schema, which will be an extensible information model and schema for general policy representation, according to the specified framework. This core information model and schema will then be extended to provide a mechanism for traffic management and QoS provision.

The objectives of the policy workgroup are the identification of representative use cases for PBNM, the definition of a framework for intra-domain policy definition and administration, the specification of an information model based on the CIM/DEN model, for the representation of policies. This information model will be refined for representing signalled and provisioned QoS. The policy framework workgroup will also co-operate with other IETF workgroups that have policy-related work.

The policy workgroup will not try to define a specific protocol for policy based management, or include schema attributes or classes that are vendor-specific. However the schema that is defined could be extended by vendors.

6.2.1.4.2.1 PBNM Architecture

The IETF policy workgroup has specified the following components for a policy based network management architecture:

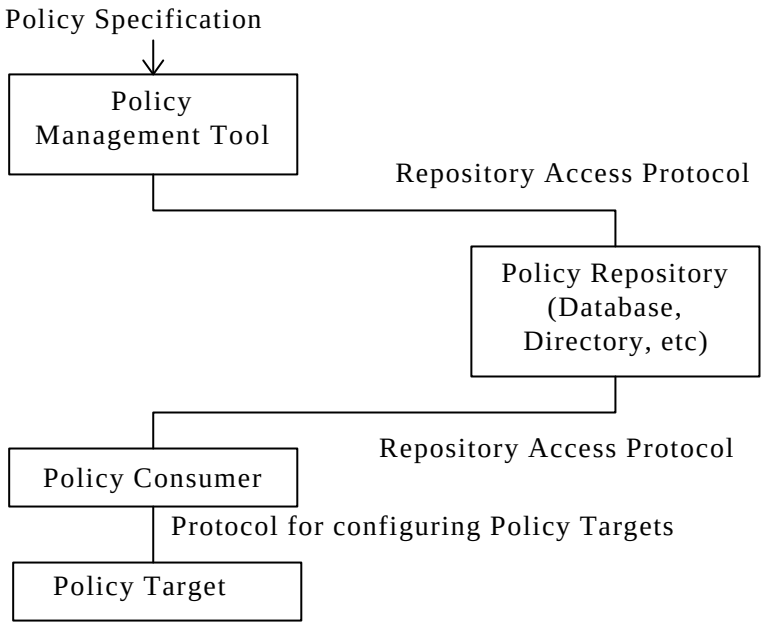

Figure 57 - IETF Policy Based Network Management Architecture

This architecture identifies four functional components

Policy Management Tool

With this tool the administrator can define new policies for the system, edit existing ones, or simply view the policies which exist in the network. The definition of policies may be done in a higher level language, which offers abstractions to the administrator. In this case, the tool also performs a translation of these high level policies, to a set of policy rules that can be interpreted by the policy consumer. The information model and schema, also defined by the IETF, specify the format of the policy rules.

Before storing a new policy, the policy management tool must check if the policy is syntactically correct. A global conflict mechanism also exists, to check if the new policy conflicts with other existing ones. This can happen when two policies have conditions that can be true at the same time, while the actions they trigger are contradicting. This conflict check is not complete, as it can check only statical conditions. For a rule that contains time-based or more dynamical conditions a run-time check is required. This work is done in another component.

Policy Repository

The policy repository is used for the storage of policies, after they have been defined and validated by the policy management tool. There should also exist a protocol that enables read and write access to the directory.

The general framework does not require a specific implementation for the policy repository, or the repository access protocol. However, the current work of the IETF includes the use of a directory server, with the LDAPv3 protocol for access. The policies are stored in a format compliant with the CIM (Common Information Standard) specification by the DMTF.

Policy Consumer

The policy consumer is the component that retrieves policies from the directory, evaluates them and sends the necessary commands to the policy target. Of course the evaluation of certain policies may also be done on the policy target, if it is capable of understanding policies. The distribution of the policy evaluation process between the consumer and the target also depends on the conditions of a policy. For example if the target is policy-aware and the policy condition concerns only the specific device, the evaluation can be done on the policy target. If however the condition is time-based or depends on the overall state of the network, then the evaluation should be done by the consumer.

Additionally the policy consumer performs a local conflict detection check, checking only those devices that are controlled by the specific consumer. The policy consumer also checks if the resources needed for a specific policy are available in all the controlled devices

A policy consumer can be integrated into a policy target, or it may exist in another device. However there is a need for at least one consumer in every administrative domain.

Policy Target

The policy target is the managed device, where the policy is finally enforced. The policy target may be policy-aware (able to interpret policy) or policy-unaware. In the case that the target does not understand the policy information model, then the interpretation of the policy must be done on a higher level, either by the policy consumer or by a policy-aware proxy. The policy will be interpreted by the proxy or consumer, which will then have to communicate with the target using a mechanism supported by the device, i.e. use SNMP to access the MIB of the device, or even use telnet commands. Policy-aware devices may also vary, according to the level of support they have for policies. For example a policy-aware may be able to interpret only a portion of the information model. So some policies can be interpreted directly by the policy target, while others will have to be interpreted by the policy consumer.

There is a need for a transport protocol, used for the communication between the policy consumer and the policy target, so that the consumer can send policy rules or configuration information to the target, or read configuration and state information from the device. There is not a requirement for a specific protocol for this operation, however the COPS protocol, defined by the IETF Resource Allocation Protocol (rap) workgroup, is becoming the standard.

6.2.1.4.2.2 Policy Representation

The language used for the specification of policies has a format of “if (condition) then (action)”. It is possible to combine multiple conditions using and/or.

Policy rules are mapped to policy classes, defined in the policy information model. This means that for all supported actions and conditions the corresponding classes should also exist in the information model. The current model defines two hierarchies of object classes: structural classes representing policy information and control of policies, and association classes that indicate how instances of the structural classes are related to each other. The current information model contains classes that support only time-based and packet-based (based on packet header fields) conditions.

There is also a schema that defines the mapping of the policy classes to a directory that uses LDAPv3 as its access protocol. The main issue here is the mapping of the relationship hierarchy between the classes, to a form suitable for storage in a directory. In order to avoid implementation incompatibilities, a common core schema is defined, for the mapping of policies to a directory. This schema is based on the CIM specifications and is being developed by the IETF policy workgroup in association with the DMTF SLA workgroup.

The policy information model is being extended, to add support for new features. There is an extension for the use of a policy based management system for admission control during QoS signalling. There is also an information model for reporting device capabilities, so that the policy consumer can determine the capabilities of the policy target.

We should mention that the work on the policy information model and schema is still under progress and no final results exist in the form of a standard.

6.2.1.4.2.3 Policy Documents

The policy workgroup has not provided any RFCs, however there exist the following Internet drafts:

- Policy Framework LDAP Core Schema
- Policy Core Information Model - Version 1 Specification

- QoS Policy Schema
- Policy Framework QoS Information Model
- Information Model for Describing Network Device QoS Mechanisms
- Policy Terminology

6.2.1.4.3 IETF Resource Allocation Protocol workgroup

An important goal of policy based network management is the provision of Quality of Service. Because of that, the co-operation between the policy based management system and QoS provisioning protocols is required. The IETF Resource Allocation Protocol (RAP) workgroup is working to provide a scalable policy control model for the Resource Reservation Protocol (RSVP). The purpose of this workgroup is the definition of a protocol for use among RSVP-capable network nodes and policy servers. For this purpose it will also be required to provide extensions to RSVP for policy control.

This workgroup has specified the Common Open Policy Service (COPS) protocol for the communication between a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP). There is also a definition for the usage of COPS for policy provisioning transactions. This workgroup will also deal with the object syntax for carrying policy information.

QoS provision is done using policy data elements contained in RSVP messages. These data elements are processed by the policy based management system, to determine if the reservation request will be granted.

The Resource Allocation Protocol Workgroup has provided the following RFCs :

- Signalled Preemption Priority Policy Element (RFC 2751)
- Identity Representation for RSVP (RFC 2752)
- A Framework for Policy-based Admission Control (RFC 2753)
- The COPS (Common Open Policy Service) Protocol (RFC 2748)
- COPS usage for RSVP (RFC 2749)
- RSVP Extensions for Policy Control (RFC 2750)
- Application and Sub Application Identity Policy Element for Use with RSVP (RFC 2872)

There are also the following Internet drafts

- Definitions of Managed Objects for Common Open Policy Service (COPS) Protocol Clients
- COPS Usage for Policy Provisioning
- Structure of Policy Provisioning Information (SPPI)
- Framework Policy Information Base
- Identity Representation for RSVP
- Signalled Preemption Priority Policy Element
- The Policy Device Auxiliary MIB

6.2.1.4.4 Discussion on DMTF Policy Based Network Management

DMTF work on Policy Based Network management is mainly focused on the representation of policies and the integration of the policy information model and schema with the CIM standard, which has also been specified by the DMTF. In the IETF, the Policy Framework workgroup is working on a policy information model, in co-operation with the DMTF, while also defining a framework for a PBNM system and identifying the necessary functional components. The Resource Allocation Protocol workgroup has defined the COPS protocol, which can be used for the communication between a Policy Decision Point and a Policy Enforcement Point, and it has also defined a framework for the use of COPS for RSVP admission control.

The policies that can be defined are limited by the current information model, which includes only classes for the representation of policy conditions based only on time and on packet headers. In the case that we need to specify different types of conditions, like conditions based on the status of the node or the network, it is required to extend the core information model. However to enforce such policies, we may also need to modify the information model. This can happen, because a policy based on the status of a certain link for example, may be either too complex (contain multiple conditions), thus requiring more processing time for its evaluation, or too much device-specific, reducing the reusability. In such cases, a possible solution would be to create two separate policies, with an association between them, so that the first policy (based on the link status) is evaluated by the policy server, causing the deployment of the corresponding policy to the target, according to the link status. However such associations are not possible with the existing information model.

According to the policy framework, policies are defined or modified by an administrative tool and the intervention of the administrator is always required. If we want to create more dynamic policies, to enable for example a self-configuring network, modifications may be required to the policy information model and architecture.

For the storage of policies, an LDAPv3 directory using the CIM schema is used. This solution is good when there are not many updates on the directory, while there are frequent accesses. In the case where write operations are more frequent, the performance of LDAP drops. Additionally a mechanism for communicating changes in the directory is also needed, which does not currently exist in LDAP. This may be solved by the IETF by extending LDAP with the necessary notifications Alternatively, a representation of policies in a language like XML might be more lightweight.

6.2.2 Policy Based Network Management Approaches

Policy Based Network Management is a relatively new field with the first commercial PBNM tools making their appearance in the IT market. The main objective of these tools is the management of IP networks, namely the configuration of these networks and also the provision of Quality of Service. In this document we will make an overview of these commercial PBNM products and we will identify any possible limitations, concerning the possible use of such tools for the management of an active network.

6.2.2.1 Policy Based Network Management Tools

In all the commercial PBNM products, policy definition is done through a graphical user interface. First the administrator selects the device or group of devices to which the policy will be applied. Then he can select from a menu of supported condition types and supply the necessary parameters. Then he can select from the supported actions. The PBNM tool identifies the selected devices, to check their capabilities and present to the user menus with only those conditions and actions that are supported by all the devices. The policy defined by the user has an “if (condition) then (action)” syntax. This kind of format is also used by the IETF.

The administrator can also create “roles” for network devices or interfaces and associate a set of policies with each role, so that when a new device is introduced in the network it can be assigned an existing role. In this way the configuration of the network is made easier.

The conditions supported by the commercial tools mainly fall into two large categories: time-based conditions and packet-based conditions (based on the packet header). The packet conditions may be specified on various layers. Layer 1 conditions can be based on a physical port or a virtual/logical port. Layer 2 conditions can include source/destination VLAN/MAC addresses. Layer 3 conditions examine the source or destination. IP address, group of addresses or subnet, the protocol type, or DiffServ/TOS marking. Layer 4 conditions involve source/destination TCP/UDP ports. Finally we can have Layer 5, or higher conditions based on HTTP/FTP headers or on specific applications. Of course not all PBNM tools support the full range of these conditions. Additionally the managed devices must have the corresponding capabilities to enforce these policies.

The actions that can be taken, when the conditions of a policy are satisfied, include the configuration of the queuing mechanism on the router, traffic coloring, denial of service to the specific flow, prioritization of traffic. As with policy conditions, the actions also depend on the capabilities of the managed devices.

After the definition of a new policy, the policy is checked for possible conflicts with other existing policies. If there is no such problem, it is stored in the policy repository. For the implementation of the policy repository, the commercial PBNM tools have adopted various solutions. Some of them use a directory, accessed using the LDAP protocol, others use a database or a flat file. In the case where a directory is used, the PBNM tools follow different approaches in the format and the use of the directory.

Another use of Policy Based Network Management is the provision of Quality of Service. Some commercial tools have support for the RSVP protocol and for DiffServ, in order to deliver QoS.

The format of the policy rules, that is the format in which the policies are stored in the directory and the format in which policies are transferred to the target devices, is also not common among the different products. Some of the tools use the CIM schema for storage of the rules in the directory. For the transport of policy rules, when the COPS protocol is used, there is also a corresponding information model defined by the IETF. In general, there is not much information about such specific implementation details.

For the configuration of the network devices according to the specified policies, a variety of protocols are used. Most of the PBNM developers use the COPS protocol for communication with the devices. If a device does not support COPS, a COPS proxy agent is used for the configuration. SNMP is also used for device configuration, while in some cases CLI (Command-line interface) commands are still used. However COPS is the emerging standard and in the future all tools will support it.

Table 3 summarises the features of the existing policy based managemet tools.

	Database support	Configuration Protocols	Device support	Policy conditions	Advantages	Disadvantages
Allot	LDAP	COPS, CLI	Allot, Cisco	Layer 1, 3, 4, 5+, time based	Active feedback mechanism	Proprietary.
Cisco	Flat-file	CLI	Cisco, Hitachi, Lucent	Layer 3, 4	Can import the topology from Cisco application .Define policies on a per-interface basis	No CIM or LDAP. No COPS support yet.No time-based conditions
Extreme	LDAP	COPS	Cisco and other	Layer 1, 2 (partially), 3 (partially), 4	Supports integration with a WinNT environment	No device discovery. No time-based conditions.
HP	?	COPS	HP switches, Cisco, Intel, Nortel routers	Layer 1, 2 (no VLAN) 3, 4, 5+ and time based	RSVP capabilities, and use of COPS for policy provision	No LDAP support. No device discovery. No queuing configuration.
IPHighway	?	COPS and CLI	Cisco routers and any COPS device	Layer 1, 3, 4 and time based	RSVP capabilities	Focused on COPS devices
Lucent	LDAP (CIM schema)	LDAP, SNMP, COPS (but not used)	Cisco LAN routers, Lucent switches.	Layer 3,4, time based conditions	Extended use of LDAP. Can define policies based on particular hosts and users, can translate a user's MAC address to an IP-based policy.	Limited set of conditions and actions. No COPS application.
Nortel	Oracle database	SNMP, CLI	Cisco and Nortel	Layers 1-4, time-based	No special advantage	No LDAP and COPS
Orchestream	Oracle database	SNMP, CLI and TACACS+	Cisco, Lucent, Xedia	Layer 3, 4 and time based	Network topology discovery. Special emphasis on Diffserv	No COPS. No support for Layer 2 conditions.
Spectrum	LDAP	CLI,SNMP	Cabletron switches and routers	Layers 1-4 and time-based	Largest range of conditions. Significant IP and IPX support. Latest CIM specifications. Topology aware	Lack of multidevice management. No COPS support yet.

Table 3 - Summary of features of existing policy based management tools.

6.2.2.1.1 Conclusions on Existing Policy Based Management Tools

The existing policy based management tools are commercial products and consequently are targeted at the management of existing business networks. This means that their primary concern is the delivery of a functional PBNM system that can be used by customers, without necessarily focusing on specific standards or trying to provide an open solution. We can observe that almost all the developers of PBNM systems are also manufacturers of network devices, whose main aim is to provide a good management platform for their own products. Even in the case where support for multivendor equipment exists, some additional features offered are proprietary. The use of COPS will make it easier to support equipment from different vendors, since a set of common capabilities will be standardized.

The definition and the modification of the policies require the manual intervention of the administrator. The system administrator should monitor the status of the network, using a network management application and then use the PBNM tool to define or modify policies, according to the status of the network. Using the commercial tools it is not possible to provide a higher degree of automation in the configuration of the network, where the intervention of the administrator would be less frequent.

The conditions available by all PBNM tools belong to two main categories: time-based and packet-based. It is not possible to define policies based on more dynamic conditions, like the status of the node or the network. In this way the policies that can be defined are less flexible, since they cannot adapt to the status of the network.

The policy conditions and actions available by each tool are dependent on the capabilities of the supported devices. This means that the set of supported actions and conditions is limited to those available by the supported devices and it is not possible to add new features without modification of the PBNM application. So if these tools were to be used for the management of a network that introduces new managed resources, for example if we wanted to manage the resources allocated to a specific execution environment, it would not be possible to take advantage of the new resources.

The commercial Policy Based Management tools are focused on the management of business IP networks. Using these tools it is possible to manage the configuration of a network and provide some Quality of Service. However these products are limited to the needs of current IP networks and they would require modifications in order to manage a network with additional features, i.e. an active network. Another drawback is the fact that each tool supports a specific set of hardware devices and the introduction of a new device is not feasible, if it does not belong to the list of supported devices.

6.2.2.2 Policy Based Network Management Protocols

The protocols we consider here take place in the well-known three-tier architecture for Policy Based Network Management (PBNM) Systems. On top of the architecture we have the Information Storage Entity (ISE), a database or a directory, which is linked to the user console and may be also to other databases constituting a distributed directory. The ISE allows the user the deployment of management policies. One level below the ISE we have the Decision Making Entity (DME). The ISE and the DME interact themselves by means of a directory/database protocol. Representative examples of such a protocol are LDAP and SQL. In the present document we pay attention only to LDAP because the push it has received from the Internet community. Finally on the bottom of the architecture we have the Enforcement Entity (EE). The EE interacts with the DME by means of a policy protocol properly said. Among several alternatives (SNMP, COPS, CLI, HTTP), COPS is by far the most appropriate and hence is the one that has been considered in the document.

6.2.2.2.1 Lightweight Directory Access Protocol (LDAP)

In 1988 the CCITT created the X.500 standard which organises directory entries in a hierarchical name space capable of supporting large amounts of information. It also defines powerful mechanisms to make retrieving of information easier. The communication between the directory client and the directory server is supported on the full OSI stack; specifically, the application level protocol is called Directory Access Protocol (DAP).

The above characteristics make X.500 a powerful directory support service but at the same time requiring resources that quite often are unavailable in many small environments. Therefore a more portable alternative was required. Specifically, attention was paid on the DAP; a lightweight version of it was proposed and termed LDAP. The basis of the complexity reduction of LDAP versus DAP is that it uses TCP/IP instead of the full OSI protocol stack and also omits some X.500 operations and features.

LDAP has been evolved through three versions up to now. The most widespread used was version 2 which is defined in [1] and completed by [2],[3],[4],[5]. LDAP version 3 is a proposed standard and is described in [6] and its accompanying RFCs [7],[8],[9],[10]. An API in C language for the LDAP client, covered by [11], has become a de facto standard. Finally it is also worth mentioning the reference [12] for implementation details of LDAP.

6.2.2.2.1.1 The LDAP client-server model

The LDAP is a communications protocol. That is, it defines the transport and format of messages issued by a LDAP client to a LDAP server to retrieve, store, modify information in a X.500 structured directory. The LDAP client interacts with the LDAP server through the set of primitives, that will be highlighted hereafter, on top of the TCP/IP protocol. For the LDAP server to interact with the directory there are two alternatives as depicted in the figure. The first alternative is based on the use of a conventional X.500 server and the second alternative is through direct interaction with the directory. The first approach makes the LDAP server simpler than the second because it acts only like a gateway but the drawback is that it requires a X.500 server that is not always available. But in both cases only the LDAP subset of the X.500 directory service capabilities is used. Therefore the implementation of the LDAP server is transparent to the LDAP client

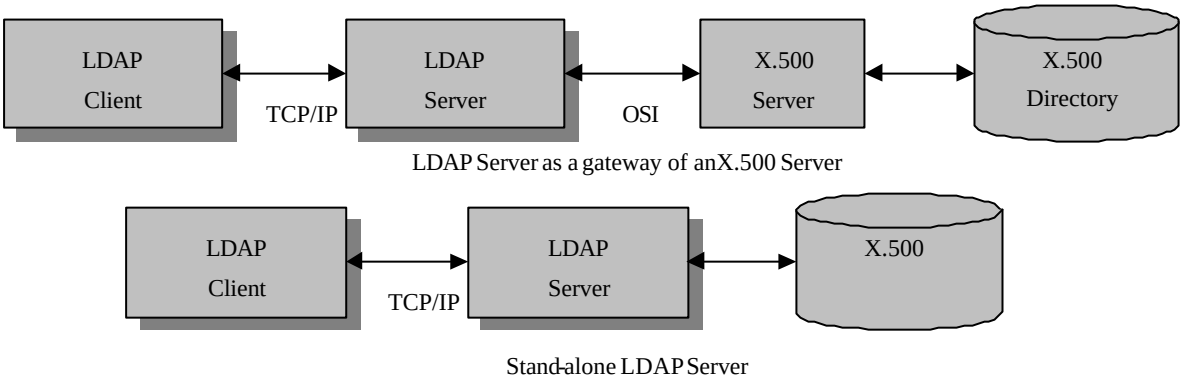

Figure 58 - Different implementations of the LDAP Server

6.2.2.2.1.2 The LDAP architecture

LDAP defines the syntax and semantics of the information exchanged between the client and the server. This information is structured in messages specifying the operations requested by a client (search, modify, delete, etc) and the responses from the server. Also, LDAP covers security related issues. All of these aspects constitute the LDAP architecture which is described in terms of a model consisting in four viewpoints, namely, Information Model, Naming Model, Functional Model and Security Model

6.2.2.2.1.3 LDAP Information Model

The information model in LDAP is object oriented. Object classes are specified by what is called a Schema. The schema defines object classes and hence their inheritance relationships with other object classes and also the position of each class in the Directory Information Tree (DIT). The attributes characterising each class are also specified. In order to ensure interoperability between different servers, many common schema must be standardised (is the same case as in MIBs of management systems).

The attributes of the above mentioned classes must have specific syntax. LDAP specifies many of these syntax as well as the behaviour of the attributes during the search process (i.e. lexicographic ordering enabled or disabled, case sensitivity, spaces/dashes ignored or not).

The information following that model is physically stored in the directory. The information storage unit is called an Entry. The entry can consist in one or many object classes.

6.2.2.2.1.4 LDAP Naming Model

The LDAP naming model defines how entries are identified and organised. Entries are organised in a tree-like structure called the Directory Information Tree (DIT). This tree allows each entry to be characterised by its Distinguished Name (DN) together with the concatenation of Relative Distinguished Names (RDN) encountered from each specific entry to the root of the. The RDN for each entry is the pair <attribute-name = attribute-value> or even the sequence of several of such pairs.

The scheme for positioning entries into the DIT may follow any kind of criteria. For instance, geographical, organisational and a combination of the two may be the basis.

An LDAP directory can be distributed. This means that a particular server must not store the entire DIT. Therefore, the LDAP servers need to be linked in some way in order to form a distributed directory that contains the entire DIT. This is accomplished by means of what is called Referrals. A referral is like a pointer available in a given server, pointing to another server that contains a part of the DIT. The information of the referral is contained in the value of an attribute (ref) defined in an object class (referral). This value adopts the format of an URL (the URL of the required LDAP server). When an application uses LDAP to request information from a server that only contains a referral for such information, the LDAP URL for that information is passed to the client, which takes the responsibility to contact the new server to retrieve such information. By the way, this is unlike both DCE and X.500 where the server takes such responsibility.

6.2.2.2.1.5 LDAP Functional Model

It defines the commands that can be issued from the LDAP client to the server. These commands can be grouped in three categories: Query, Update and Authentication. Commands belonging to any of these categories point to one or several entries of the DIT. The selection mechanism foreseen in LDAP is like the one used in CMIP but even more flexible. To perform a search the following parameters must be specified:

- Base object: the DN of the DIT from which the search is started
- Scope: specifies the depth of the search from the Base. It can be specified as baseObject (only the base object is examined), singleLevel (only the child of the base object is examined) and wholeSubtree.
- Filter: Boolean combinations of attribute value assertions (AVAs) to refine the search process
- Attributes to return: Selects the attributes to be inspected
- Limits: specifies upperbounds for the time and size of the search.

6.2.2.2.1.6 LDAP Security Model

The LDAP security model covers authentication, integrity and confidentiality aspects in different levels that can be negotiated at the establishment of the connection between the client and the server.

6.2.2.2.2 The Common Open Policy Service (COPS) Protocol

The IETF Resource Allocation Protocol (RAP) Working Group has developed COPS as a policy protocol for use in Policy Based Network Management (PBNM) systems. It was developed in contrast to traditional network management protocols like SNMP, which was found incapable to efficiently support PBNM. The COPS protocol stack can be conceptually divided into three distinct layers: the base protocol, the client-type usage directives and the policy data representation.

The base protocol defines the communication mechanism for facilitating the exchange of policy information between a Policy Decision Point (PDP) and its associated Policy Enforcement Points (PEP). The PEP (each PEP) takes the role of client and the PDP is the server. Unlike SNMP, COPS in based on TCP connections and is a stateful protocol, that basically means that the server keeps track of the state of the client (or clients), reacting appropriately, in an unsolicited manner, when necessary. COPS base protocol is defined in [13].

The COPS built-in concept of client-types leaves room for adding a second layer of client-specific directives and expansions, namely COPS-RSVP [14], COPS-PR [15] and COPS-MPLS [16]. COPS-RSVP is foreseen when using an outsourcing PBNM model and the RSVP protocol is used as the signalling mechanism. On the other hand, COPS-PR is foreseen for use in a provisioning PBNM model. COPS-MPLS is also for a provisioning model, managing an MPLS network and its traffic engineering functionality. The main differences are due to the different messages between the client and the server and also to the different information carried by these messages. For instance COPS-RSVP may re-use policy data objects defined in RSVP whilst COPS-PR requires an ad-hoc Policy Information Base (PIB).

As mentioned before, COPS is quite flexible in the sense that it can support objects defined in other contexts or its specific ones. As described in [15], each client supports a non-overlapping and independent PIB. However, some policy rule classes are common to all client types and replicated in each. Reference [17] presents the PIB classes that are common to all clients that provision policy, using COPS for provisioning. In this context is also relevant to mention a Management Information Base (MIB) containing objects for managing COPS client devices [18],[19], where the relationship between device interfaces and policy combinations are also included.

6.2.2.2.2.1 The COPS base protocol

The COPS protocol has the following characteristics:

- The protocol employs a client/server model where the PEP sends requests, updates, and deletes to the remote PDP and the PDP returns decisions back to the PEP.
- The protocol uses TCP as its transport protocol for reliable exchange of messages between policy clients and a server. Therefore, no additional mechanisms are necessary for reliable communication between a server and its clients.
- The protocol is extensible in that it is designed to leverage off self-identifying objects and can support diverse client specific information without requiring modifications to the COPS protocol itself.
- COPS provides message level security for authentication, replay protection, and message integrity. COPS can also reuse existing protocols for security such as IPSEC, TLS or CMS to authenticate and secure the channel between the PEP and the PDP.
- The protocol is stateful in two main aspects: (1) Request/Decision state is shared between client and server and (2) State from various events (Request/Decision pairs) may be inter-associated. By (1) we mean that requests from the client PEP are installed or remembered by the remote PDP until they are explicitly deleted by the PEP. At the same time, Decisions from the remote PDP can be generated asynchronously at any time for a currently installed request state. By (2) we mean that the server may respond to new queries differently because of previously installed Request/Decision state(s) that are related. Additionally, the protocol is stateful in that it allows the server to push configuration information to the client, and then allows the server to remove such state from the client when it is no longer applicable.

6.2.2.2.2.2 The Protocol Information Model

The protocol information model is defined through describing the message formats and objects exchanged between the PEP and remote PDP. Each COPS message consists of the COPS header followed by a number of typed objects. All the objects follow the same object format; each object consists of one or more 32-bit words with a four-octet header.

6.2.2.2.2.3 The Protocol Functional Model

The protocol functional model is defined through basic messages exchanged between a PEP and a remote PDP as well as their contents.

6.2.2.2.2.4 Security Considerations

The COPS protocol provides an Integrity object that can achieve authentication, message integrity, and replay prevention. All COPS implementations must support the COPS Integrity object and its mechanisms. To ensure the client (PEP) is communicating with the correct policy server (PDP) requires authentication of the PEP and PDP using a shared secret, and consistent proof that the connection remains valid. The shared secret minimally requires manual configuration of keys (identified by a Key ID) shared between the PEP and its PDP. The key is used in conjunction with the contents of a COPS message to calculate a message digest that is part of the Integrity object. The Integrity object is then used to validate all COPS messages sent over the TCP connection between a PEP and PDP.

The COPS Integrity object also provides sequence numbers to avoid replay attacks. The PDP chooses the initial sequence number for the PEP and the PEP chooses the initial sequence number for the PDP. These initial numbers are then incremented with each successive message sent over the connection in the corresponding direction. The initial sequence numbers should be chosen such that they are monotonically increasing and never repeat for a particular key.

Security between the client (PEP) and server (PDP) may be provided by IP Security (IPSEC). Transport Layer Security (TLS) may be used for both connection-level validation and privacy [20] as well as Cryptographic Message Syntax (CMS) [21].

6.3 INITIAL SUMMARY OF ACTIVE NETWORK BASED APPROACHES TO NETWORK MANAGEMENT

This section identifies the benefits of active networking based upon the limitations given previously in section 7.2. We also identify the potential problems or issues that might arise when we use active networks for network management, e.g. managing networks where new protocols can be deployed on the fly; issues of security; requiring management of the networks as well as management of the active nodes etc.

6.3.1 Virtual Active Private Networks

The Virtual Active Network (VAN) is a key concept in the management architecture for active networks, which has been proposed in [34]. The VAN is the entity that the active network provider makes available to a service provider. Consisting of virtual active nodes interconnected by virtual links, the VAN provides the platform on which a service provider installs a service.

The main ideas of this concept, including some FAIN specific modifications, are briefly described in the following paragraphs.

6.3.1.1 Actors

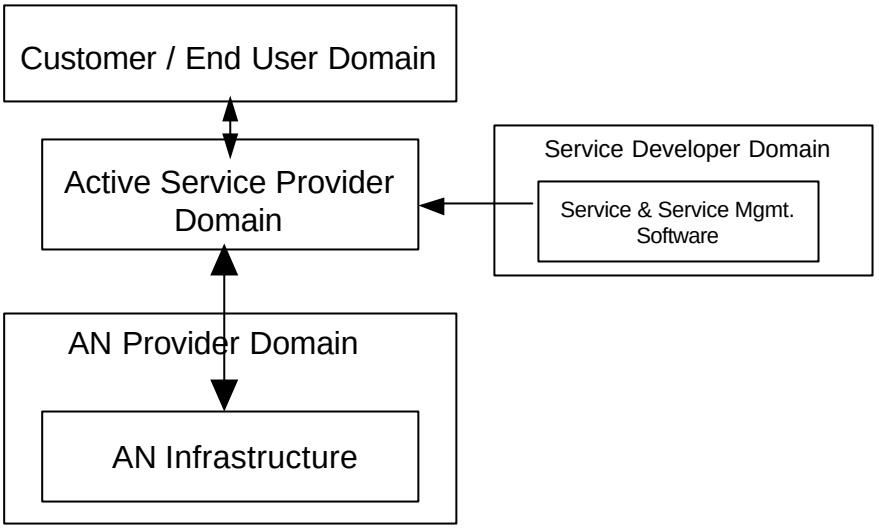

Figure 59 - Actors for Service Creation in Active Networks

In the context of service creation on active networks, we consider the following four actors (Figure).

- An *active network provider* owns the active network infrastructure. The active network provider furnishes an environment on which the service provider can independently install and manage a service.
- A *service provider* offers a service to a customer. The service runs on an active network. The service provider also manages this service. The service implementation may be bought from a service developer.
- A *service developer* is an actor that develops service implementations, e.g. a software house. The implementation may be sold to a service provider.
- A *customer* uses a service made available by a service provider.

6.3.1.2 VAN Management

The Virtual Active Network concept aims at providing a powerful generic service abstraction to the service providers. A Virtual Active Network consists of virtual active nodes interconnected by virtual links. Virtual active nodes are also called Execution Environments (EEs), following the terminology of the AN working group. A virtual active node has resources attached to it in form of processing time and memory, provided by the underlying active networking platform. Similarly, a virtual link has bandwidth allocated to it. A single physical active node can run several virtual active nodes belonging to different VANs, and a single physical network link can support several virtual links for different VANs [35].

The VAN is a generic service abstraction. In this context, generic means that the VAN is not aware of the services running on top of it. As a result, the management system of the active network provider is not involved in the management of the service running on top of a VAN. It solely manages and supervises the VANs. The VAN is the only object that is negotiated between the active network provider and the service provider. As a consequence, management complexity is reduced from an active network provider's point of view.

The VAN is also the entity of isolation between services. The operating system of a physical active node must provide means to efficiently allocate resources to an EE (virtual active node), and to control and enforce the consumption of the allocated resources according to the contract between service provider and active network provider.

6.3.1.3 Implications on FAIN Node Architecture

The VAN concept makes a clear separation between management EE that works on the management plane and service provider EE, which works on the data transfer as well as on the signalling/control plane. Only these two types of EE exist in the VAN architecture.

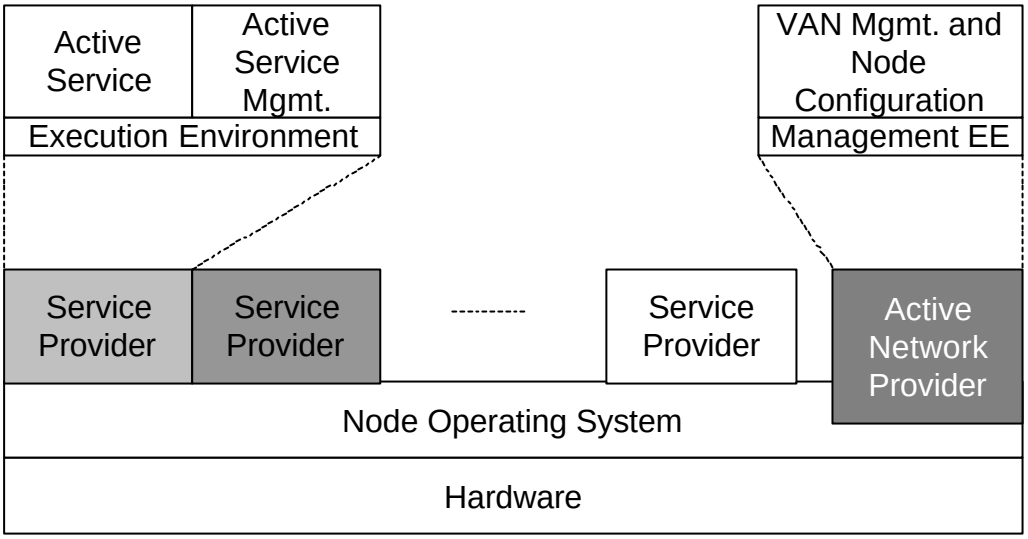

Figure 60 - Active Network Node Architecure

The tasks of the management EE are limited to node configuration and the management of VANs in the active network provider's domain. Note that in this context VAN management means the creation, modification, monitoring, and termination of virtual active networks. The management EE is **not** concerned with the management of active services running in the VANs.

In the VAN architecture a service and the corresponding service management run in the same instantiation of a service provider EE.

The VAN concept strictly isolates different VANs from each other. A misbehaving service provider can not affect other service providers.

6.3.1.4 Implications on Node Operating System

In order to provide VANs, the node operating system, handles resource allocation for the different EEs. The resources are allocated in a way to fulfil the contract between service provider and network provider. The node operating system isolates the EEs from each other, and supervises resource usage for each EE.

Management EE and customer EE have totally different requirements for the underlying node operating system. A Management EE which works on the management plane may need to provide a rich interface to network management software. We may think of the management EE as a Java Virtual machine. The service provider EEs on the other hand need to work on the data path in order to perform computation on active packets. As a consequence performance is of major importance to the service provider EE. We think that it would be useful to run management and service provider EEs on separate hardware and node operating systems.

6.3.1.5 Benefits for FAIN

Generic Abstraction

The VAN abstraction is service-independent. A provider is not concerned with the services and the corresponding management running on VANs. The VAN is the only object that a provider has to negotiate with a service provider. It is also the only entity according to which active network resources have to be allocated and enforced in order to isolate service providers from each other.

Clear Separation between Execution Environments

The tasks of the two different types of execution environments (EE) are clearly separated. The management EE works on the management plane, whereas the service provider EE works on the data transfer and on the signalling/control planes. The management EE has privileged access to the underlying node operating system.

Flexible Service Abstraction

The service provider can flexibly select the VAN topology. Between the two extreme cases – a VAN spanning over the all active network nodes of a network provider and a VAN consisting of one virtual link through the physical network – any topology is possible.

Services Managed by Service Providers

Through the introduction of the VAN concept, services are isolated from each other. This enables services that are managed by the service provider, because even a misbehaving service does not affect other services. As a consequence, management complexity is reduced from an active network provider's point of view.

Fine-grained Resource Allocation

Since a VAN is a collection of computation, communication, and storage resources on active nodes, a service provider can fine-tune the usage of each resource on each node separately.

6.3.1.6 Challenges

Isolation of VANs versus Performance

An important challenge in active networking is to trade-off security versus performance. For security and in order to guarantee the quality of the VAN, it is important to strictly isolate the VANs. However, it has to be investigated whether weakening this isolation for certain services would save resource usage.

Interaction of Management and Customer Execution Environments

As stated earlier in this report, the management execution environment (EE) and the customer EE have totally different requirements. The management EE works on the management plane, whereas the customer EE are supposed to work in the data transfer and signalling/control plane. Therefore, we think that the EEs should run on separate environments (hardware, operating system). However signalling between the two types of EEs is required. It remains to be investigated, how this could be done best.

6.3.2 Existing Policy Based Network Management Approaches

In order to make an efficient distribution of the active network management tasks between users and operators, it is necessary to make the most of the active nodes computational capabilities to allow the downloaded applications to accomplish control functions on the data flows.

The policy-based network management architectures fit this objective best. They permit active code to establish its own management policies while still maintaining a suitable control level over the resources. Enhanced mechanisms are provided to assure the policies defined by applications do not jeopardize the operator-imposed management policies.

Active networks go further since they facilitate the development of "elastic" architectures which are able to increase the management functionalities by means of code downloads into the active nodes, and all without service interruption. In this way system flexibility grows as new types of policies can be defined to adapt the management system to evolving requirements of such a dynamic environment.

In this section we demonstrate the "elastic" management architecture for the policy-based network management for active networks, especially in the scope of security, where several aspects of interest for the FAIN project are collected.

6.3.2.1 GENERAL ARCHITECTURE COMPONENTS DISTRIBUTION

The policy-based management system architecture defined by IETF [39] distinguishes the following essential elements:

- The Policy Repository, where policies are stored.
- The Policy Consumer or policy decision point, where decisions for policies are made.
- The Policy Target or policy enforcement point, where policies are executed.
- The Policy Management Application, where conflicts are detected.

Two general approaches to distribute these components among the functional blocks building up the active node architecture are outlined in [40].

In the first one, the policy decision point and policy enforcement point are placed in the Node Operating System (NodeOS) whereas policies are stored in a database independent of the enforcement engine. Either the Execution Environments (EE) or the NodeOS can define their own resource access policies, even though the EE cannot circumvent the policies defined by the NodeOS. Including the policy decision point into the NodeOS compels the EE to inform it about access policies, since the NodeOS is responsible for scheduling of resources assigned to the EE in accordance to those policies.

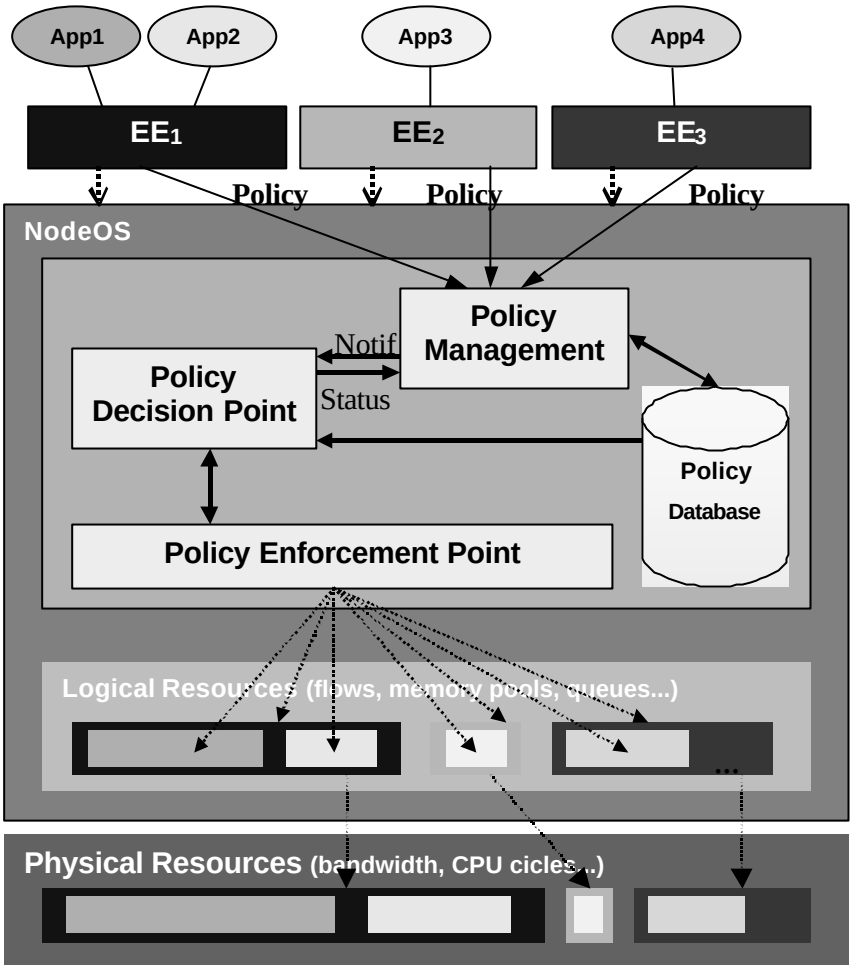

Figure 61 - “Tighten” management architecture

The second proposed model [40] relies on sharing the responsibility of the policy enforcement between the NodeOS and the EE. In order to enable the EE to enforce the policies, it has to have its own database and its “enforcement engine”.

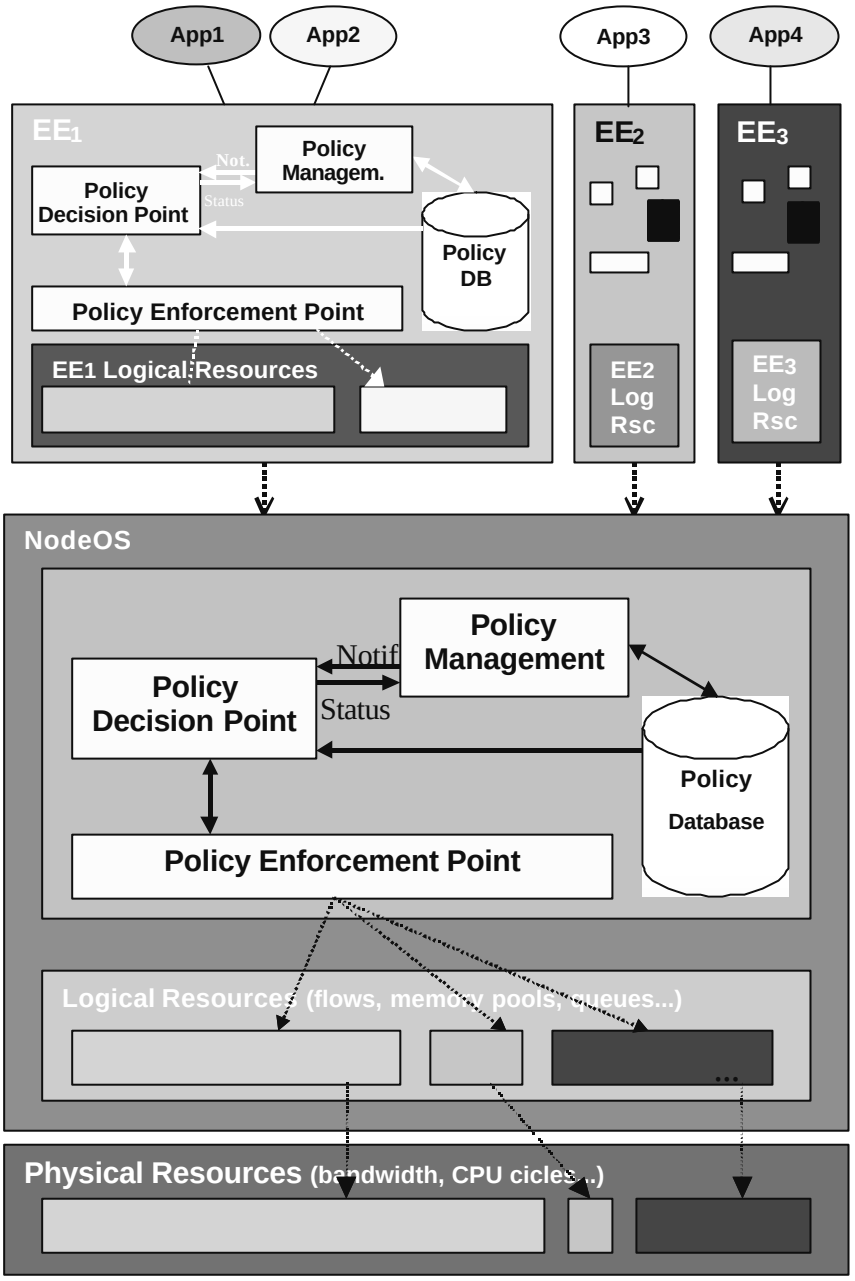

Figure 62 - "Scattered" management architecture

Thus, the policy is enforced in two levels. The EE grants access to resources and services under its control according to its own policy. When the EE requests the NodeOS resources and services, the NodeOS will enforce its access policy on the resources. To avoid as much as possible the conflict between managed policies in both levels, the NodeOS establishes a set of initial policies for the EE.

This kind of splitting up would favour the simplification of the development of the network management system and would facilitate a suitable distribution of the FCAPS[37] functionalities, e.g. according to performance requirements.

[37] Fault Configuration Accounting Performance and Security functions.

6.3.2.2 ELASTIC NETWORK MANAGEMENT ARCHITECTURE APPROACH

The elastic architecture approach [36] outline an open extensible solution for a management system design. It gives the downloaded code the chance to control the available resources. The aim of this strategy is to allow the applications to extend the basic management functionalities integrated within the node.

In contrast to using general policies, this approach suggests to make the most of the specific knowledge that clients have about their applications, e.g. through considering communication patterns, making feasible the installation of application-specific policies within the node. In this way, rapid decisions can be made to modify the behaviour of the local components according to the status information, which is very important in multimedia and other large distributed systems.

This solution has been developed for programmable networks which separate data paths from control paths. Hence the conclusions can be also valid for the "discrete"-approach active networks [41].

The fact of having active code managing the resources to be assigned by means of policies has been discussed and accepted in other active network fora[38], which eventually could contribute to the consensus on the essential services within an active network.

6.3.2.2.1 Establishing application-specific policies

When considering the proposal in [36], we have to take into account that the model analised in the document doesn't distinguish between "policy enforcement point" and "policy data". On the contrary, the policy is embedded into the code itself.

The concrete mechanism to establish an application-specific management policy lies in the injection of pieces of code by the active code itself in the core of the network management and control system. The management system is ready to accept dynamic load of management code, extending the basic functionalities.

To make feasible policy migration to different network nodes, the code must be able to send agents towards remote management systems so that the process is repeated. In this way, the policies convenient for the application could be established throughout a whole path.

In a similar vein, it would be possible to issue a new active packet from the active node. This packet would carry the policy data addressed to other active nodes in the path.

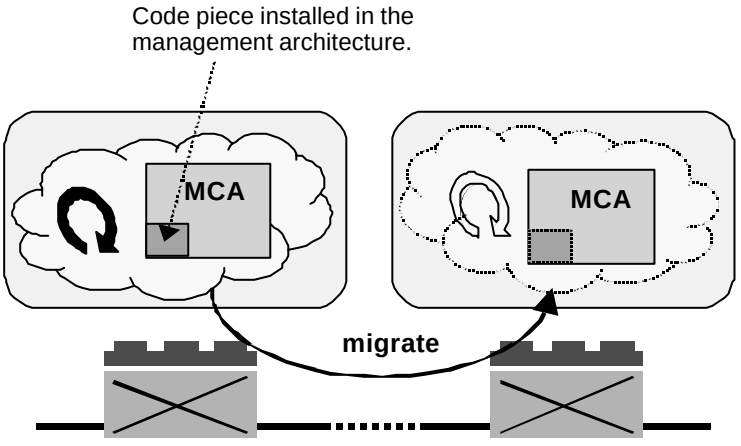

Figure 63 - Policy migration

[38]See "Active Nets Strawman Security Architecture Working Session Notes" http://www.ittc.ukans.edu/~ansecure

6.3.2.2.2 Resource partition

In order to face the QoS requirements of certain applications, the elastic architecture considers the resource reservation in advance (in the line of the Integrated Services models), by creating small virtual networks (called netlets) which offer the possibility of sharing the available resources and establishing different policy levels over them.

In this way, when an application arrives at the light virtual network, it has all the resources previously reserved available. At first, the code arriving at a netlet does not have any restriction to access its assigned resources, being allowed to handle them as required. The netlets need to be isolated enough to avoid improper behaviour of the system, e.g. excessive consumption of resources that could affect others.

6.3.2.2.3 Extending the management capabilities

An elastic architecture bears not only the inclusion of new management policies but also the dynamic loading of new code, able to provide extra functionality to the network management system. The aim is to "enable clients to program all aspects of network control in a controlled fashion. This is done by dynamically injecting code into the various entities that constitute the network control system"[38].

6.3.2.3 The Seraphim architecture

The policy-based security management system developed in the Seraphim project deals with several ideas that have already been presented, and applies them within the scope of an active network architecture.

6.3.2.3.1 Active Capabilities

Seraphim enables the extension of its security mechanisms by allowing the active code to dynamically install its own application-specific security functions. These code fragments, which are encapsulated inside active packets, have been named *active capabilities* (AC).

An AC is able to carry not only the active code, but also the security policies customised for a particular application and even, the code needed to make a policy decision. Hence, the user "can" (in some way) establish security policies in the active node.

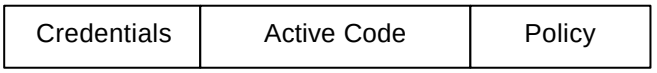

Figure 64 - Active Capability

6.3.2.3.2 Policy Framework

The active node has a framework to store, get and evaluate policies. Besides, every node has an evaluation/enforcement engine responsible for the execution of the policies loaded into the database. The elements have been distributed following the second model given above (the "scattered" architecture model).

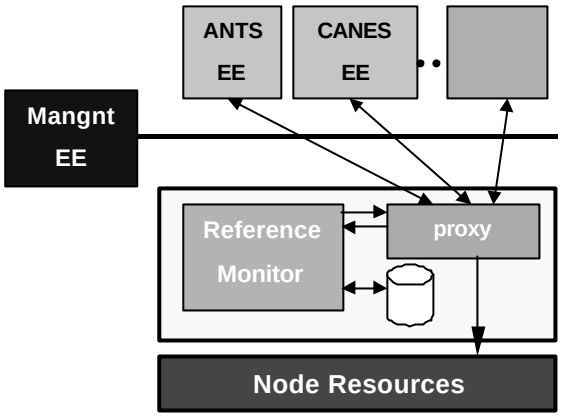

Figure 65 - Seraphim security architecture

A policy database and an enforcement engine - also known as a *reference monitor* - are included within the active node kernel. Nevertheless, the execution environment could hold its own policy database and enforcement engine, in case it would like to handle additional restrictions. In this case, the NodeOS should guarantee that the policy database within the EE is coherent with the node policy.

Again, it must be noticed that the policy framework is itself reconfigurable through the direct injection of code.

In order to make quite sure that the AC carries well-formed expressions, the system includes an AC management infrastructure, with an administrator in charge of checking that the AC cannot compromise the system operation.

6.3.2.3.3 Administration Issues

Seraphim uses a centralized administration tool. Whenever there is a policy change, the administration component attends to the matter of disseminating the new policies (ACs) to the active nodes, revoking the previous ones. In this way, the network operators can manage the essential active network capabilities in an easy way.

6.4 INITIAL ISSUES IN POLICY BASED NETWORK MANAGEMENT

We have outlined some essential questions in the development of a policy-based network management system for active networks. Flexibility and extendibility are indispensable requirements to build up a system able to adapt itself to new management mechanisms.

The need for establishing application-specific policies is one crucial area for the success of active networks for network management. It is likely that to realise this will require maintaining a database in the active node, with the aim of reducing the network management traffic. In the same way, it is convenient to include also the policy decision point into the active node. This is in accordance with the IETF work which supports policy decision points lying in the same device as the policy enforcement point when the device has enough processing capacity.

However, it should be noted that the introduction of new policies into the management system (performed by the active code) must be carefully controlled in order to avoid the execution of actions not allowed by the operator-defined policies.

Certain conflicts between both types of policies, called "modality conflicts", which arise "when several policies with modalities of opposite sign refer to the same subjects, actions and targets" [37], can be detected by means of syntactic analysis and resolved setting priorities for every policy.

However, several conflicts derive not from the policy structure but from the policy meaning. To handle these conflicts, it is necessary to specify which policies can coexist in the system. That is done in metapolicies terms. Metapolicies express semantic incompatibilities among policies [44], as for example, "there should be no policy authorising a manager to retract policies of which he is the subject". In summary, metapolicies can be a valuable tool to detect and evaluate which policies established by the active code could be included without risk in the active node management system.

Finally, the use of metapolicies should be considered as a node mechanism of protection against the attack of malicious active packets trying the establishment of forged policies. Metapolicies could be also a means of unifying the handling of application-specific policies for the whole range of management functionalities.

6.5 R25 REFERENCES

[1] RFC 1777 Lightweight Directory Access Protocol

[2] RFC 1778 A String Representation of Standard Attribute Syntaxes

[3] RFC 1779 A String Representation of Distinguished Names

[4] RFC 1959 An LDAP URL format

[5] RFC 1960 A String Representation of LDAP Search Filters

[6] RFC 2251 Lightweight Directory Access Protocol (v3)

[7] RFC 2252 Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions

[8] RFC 2253 Lightweight Directory Access Protocol (v3):UTF-8 String Representation of Distinguished Names

[9] RFC 2254 The String Representation of LDAP Search Filters

[10] RFC 2255 The LDAP URL Format

[11] RFC 1823 The LDAP Application Program Interface

[12] http://www.openldap.org/

[13] RFC 2748 The COPS (Common Open Policy Service) Protocol, January 2000

[14] RFC 2749 COPS usage for RSVP, January 2000-08-30

[15] <draft-ietf-rap-pr-04.txt> COPS Usage for Policy provisioning (COPS-PR), August 2000

[16] <draft-franr-mpls-cops-00.txt> COPS Usage for MPLS/Traffic Engineering, July 2000

[17] <draft-ietf-rap-frameworkpib-01.txt> Framework Policy Information Base, July 2000

[18] <draft-ietf-rap-pol-aux-mib-00.txt > The Policy Device Auxiliary MIB, July 2000.

[19] <draft-ietf-rap-cops-client-mib-03.txt> Definitions of Managed Objects for COPS protocol clients, May 2000

[20] <draft-jwalker-cops-tls-00.txt> COPS over TLS, May 2000

[21] <draft-jwalker-cops-cms-00.txt> CMS over COPS, May 2000

[22] Policy Framework Working Group:(http://www.ietf.org/html.charters/policycharter.html)

[23] Resource Allocation Protocol Working Group: (http://www.ietf.org/html.charters/rap-charter.html)

[24] TINA-C, www.tinac.com

[25] TINA-C, *Distributed Processing Environment*, December 1994.

[26] TINA-C, *Service Architecture*, version 5.0, 16 June 1997.

[27] TINA-C, *Network Resource Architecture*, Version 3.0, February 1997.

[28] *The Common Object Request Broker Architecture and Specification*: Revision 2.3, Object Management Group, Inc., Framingham MA., July 1998.

[29] TINA-C, TINA Object Definition Language MANUAL, version 2.3, July 1996.

[30] OMG Telecommunication Service Access and Subscription, telecom/00-02-02, www.omg.org

[31] R.O. Sinnott, T. Gringel, M. Tschichholz, W. Vortisch, Supporting Service Quality Assurance via Trouble Management, Proceedings of Third International Conference on Management of Multimedia Networks and Services 2000, September 26-28, 2000, Fortaleza- Ceara, Brazil.

[32] R.O. Sinnott, D. Dragan, T. Gringel, J. Hall, M. Tschichholz, W. Vortisch, Integrated Trouble Management to Support Service Quality Assurance in a Multi-Provider Context, Proceedings of 7th International Conference on Intelligence in Services and Networks, Athens, Greece, February 2000.

[33] More information under www.fokus.gmd.de/research/cc/platin/

[34] M. Brunner, "Service Management in a Telecom Environment based on Active Network Technology", ETH Zurich, TIK-Schriftenreihe No. 34, Diss. ETH Zurich No. 13433, November, 1999.

[35] M. Brunner, R. Stadler, "Service Management in Multi-Party Active Networks", IEEE Communications Magazine, Vol. 38(3), 2000.

[36] H. Bos, "Application-Specific Policies: Beyond the Domain Boundaries". IFIP/IEEE Integrated Management Symposium, Boston, May 1999 http://www.ee.ucl.ac.uk/iliaboti/papers/

[37] M. Sloman and E. Lupu, "Policy Specification for Programmable Networks". Extended version of paper in Proceedings of First International Working Conference on Active Networks (IWAN'99), Berlin, June 1999. Ed. S. Covaci published by Springer-Verlag Lecture Notes in Computer Science.

[38] H. Bos, "Open Extensible Network Control". Journal of Network and Systems Management, Vol. 8, No.1, March 2000.

[39] H. Mahon et al, "Requirements for a Policy Management System". Internet draft, October 1999.

[40] J. Smith et al, "Activating Networks: A Progress Report". IEEE Computer, Vol. 32, No. 4, April 1999.

[41] D. Tennenhouse and D. Wetherall, "Towards aun Active Network Architecture". Computer Communication, Vol. 26, No. 2, April 1996.

[42] Z.Liu, P.Naldurg, S. Yi, T. Qian, R. H. Campbell, M. Dennis Mickunas, "An Agent Based Architecture for Supporting Application Level Security". DARPA Information Survivability Conference and Exposition. Hilton Head Island, sc, January 2000. http://choices.cs.uiuc.edu/Security/seraphim/papers/

[43] D. Marriot and M. Sloman. "Implementation of a Management Agent for Interpreting Obligation Policy". IEEE/IFIP 7th International Workshop on Distributed Systems Operations and Management (DSOM 96), L'Aquila, Italy, 28-30 October 1996.

[44] E. Lupu, "A Role-Based Framework for Distributed Systems Management". Imperial College of Science, University of London.

[45] ITU-T Recommendation M.3010 Principles for a Telecommunications management network (05/96).

7 APPENDIX B: POLICY BASED MANAGEMENT INFORMATION MODEL

7.1 FAIN POLICY BASED MANAGEMENT INFORMATION MODEL

The inheritance tree detailed here is basically that of the PCIM extensions draft [75]. However, there are some new FAIN-specific classes. More classes or properties can be defined and added to the tree, either FAIN-specific or domain specific. Basically, domain-specific extensions to the information model will be sub-classes of PolicyImplicitVariable and PolicyValue classes. However, in case all conditions, action or rules of a specific domain have a common property, new domain-specific rule, actions or conditions subclasses might be added.

7.1.1 Classes Inheritance Hierarchy

ManagedElement (abstract)

| |

+--Policy (abstract)

| |

| +---PolicySet (abstract – PCIMe)

| | |

| | +---PolicyGroup

| | |

| | +---PolicyRule

| | |

| | +--- **fainPolicyRule**

| |

| +---PolicyCondition (abstract)

| | |

| | +---PolicyTimePeriodCondition

| | |

| | +---VendorPolicyCondition

| | |

| | +---SimplePolicyCondition

| | | |

| | | +--- **fainSimplePolicyCondition**

| | +---CompoundPolicyCondition

| | |

| | +---CompoundFilterCondition

| |

| +---PolicyAction (abstract).

| | |

| | +---VendorPolicyAction

```
| | |
| |  +---SimplePolicyAction
| | |              |
| | |              +--- fainSimplePolicyAction
| |  +---CompoundPolicyAction
| |
| +---PolicyVariable (abstract )
| | |
| |  +---PolicyExplicitVariable
| | |
| |  +---PolicyImplicitVariable (abstract)
| |     |
| |      +---(subtree of more specific classes)
| +---PolicyValue (abstract )
|    |
|     +---(subtree of more specific classes)
|
+--Collection (abstract -- newly referenced)
| |
| +--PolicyRoleCollection
|
+--ManagedSystemElement (abstract)
  |
   +--LogicalElement (abstract)
     |
      +--System (abstract)
        |
         +--AdminDomain (abstract)
           |
            +---ReusablePolicyContainer (lo mismo que el PolicyRoleCollection)
            |
            +---PolicyRepository (deprecated)
```

7.1.2 Aggregation Classes Inheritance Hierarchy

```
[unrooted]
  |
  +---PolicyComponent (abstract)
  | |
  |  +---PolicySetComponent (abstract -- new - 4.3)
```

```
| | |
| |  +---PolicyGroupInPolicyGroup (moved - 4.3)
| | |
| |  +---PolicyRuleInPolicyGroup (moved - 4.3)
| | |
| |  +---PolicyGroupInPolicyRule (new - 4.3)
| | |
| |  +---PolicyRuleInPolicyRule (new - 4.3)
| |
|  +---CompoundedPolicyCondition (abstract -- new - 4.7.1)
| | |
| |  +---PolicyConditionInPolicyRule  (moved - 4.7.1)
| | |
| |  +---PolicyConditionInPolicyCondition (new - 4.7.1)
| |
|  +---PolicyRuleValidityPeriod
| |
|  +---CompoundedPolicyAction (abstract -- new - 4.7.2)
| | |
| |  +---PolicyActionInPolicyRule  (moved - 4.7.2)
| | |
| |  +---PolicyActionInPolicyAction (new - 4.7.2)
| |
|  +---PolicyVariableInSimplePolicyCondition (new - 4.8.2)
| |
|  +---PolicyValueInSimplePolicyCondition (new - 4.8.2)
|
+---Dependency (abstract)
| |
|  +---PolicyInSystem (abstract)
| | |
| |  +---PolicyGroupInSystem
| | |
| |  +---PolicyRuleInSystem
| | |
| |  +---ReusablePolicy (new - 4.2)
| | |
| |  +---PolicyConditionInPolicyRepository (deprecated - 4.2)
| | |
```

```
| | +---PolicyActionInPolicyRepository (deprecated - 4.2)
| |
| +---PolicyValueConstraintInVariable (new - 4.8)
| |
| +---PolicyRoleCollectionInSystem (new - 4.6.2)
|
+---Component (abstract)
| |
| +---SystemComponent
|   |
|   +---PolicyContainerInPolicyContainer (new - 4.2)
|   |
|   +---PolicyRepositoryInPolicyRepository (deprecated - 4.2)
|
+---MemberOfCollection (newly referenced)
  |
  +--- ElementInPolicyRoleCollection (new - 4.6.2)
```

7.1.3 FAIN Specific Classes Description

In this section a detail definition of new classes, and modifications of existing ones, in FAIN Information Model is given. To see the definition and detail description of classes defined in PCIM, PCIMe and CIM Core Model and their properties see [62] [75]and [77] respectively.

Policy (abstract)

```
NAME            Policy
DESCRIPTION     An abstract class with four properties for describing a
                     policy-related instance.
DERIVED FROM    ManagedElement
ABSTRACT        TRUE
PROPERTIES      CommonName (CN)
                PolicyKeywords[ ]
                       // Caption (inherited)
                       // Description (inherited)
```

In the FAIN project we will add some FAIN specific keywords, to be used in the PolicyKeyword property, to those suggested by the IETF [62]. These keywords will categorised some FAIN policies such as: Delegation, Fault, Monitoring, ...

PolicySet (abstract)

```
NAME            PolicySet
DESCRIPTION      An abstract class that represents a set of policies
```

```
                    that form a coherent set.  The set of contained
                    policies has a common decision strategy and a common
                    set of policy roles.  Subclasses include PolicyGroup
                         and PolicyRule.
DERIVED FROM        Policy
ABSTRACT            TRUE
PROPERTIES          PolicyDecisionStrategy
                    PolicyRoles
```

In FAIN we will extend the possible values of the PolicyDecisionStrategy property with an Atomic option. Previous possible values were: 1st matching, All matching. This will allow to easily define atomic sets of policies.

The PolicyRoles property can be used to identify the PEP that has to enforce the policy.

fainPolicyRule

```
NAME              PolicyRule
DESCRIPTION       The central class for representing the "If Condition
          then Action" semantics associated with a policy rule in FAIN.
DERIVED FROM      PolicyRule
ABSTRACT          FALSE
PROPERTIES     UserInfo
```

The fainPolicyRule adds the UserInfo property to those define by the IETF. This property is necessary to be able to check the functionality that has been delegated to that user. The UserInfo property will be a set of strings containing security-related information of that user (e.g. login and password).

fainSimplePolicyCondition

```
NAME              SimplePolicyCondition
DERIVED FROM      PolicyCondition
ABSTRACT          False
PROPERTIES        EvaluationMethod
```

The fainSimplePolicyCondition adds a new property: EvaluationMethod. This property adds more flexibility to the IETF proposal which by default uses the MATCH evaluation method, see [75].

The property will be a string whose value represent the EvaluationMethod to be applied to that simple policy condition. Possible values, as well as the necessity of this property, will be evaluated and added as the FAIN Information Model is defined.

fainSimplePolicyAction

```
NAME              SimplePolicyAction
DESCRIPTION       A subclass of PolicyAction that introduces the notion
                  of "SET variable TO value".
```

```
DERIVED FROM     PolicyAction
ABSTRACT         FALSE
PROPERTIES       EnforcementMethod
```

The philosophy of this property is exactly that of the EvaluationMethod property within the fainSimplePolicyCondition class. It aims to extend the by-default "SET variable to value" enforcement method of the IETF.

7.1.3.1 Simple Policy Example

The rule that we are going to give as an example is a very simple rule from the VPN Policy Information Model proposed by the IETF in [76].

The rule is: if "SourceIP=147.83.106.172" then ipvpnPolicyFirewallAction=1

Possible values of ipvpnPolicyFirewallAction are: Allow=0, Allow&Log=1, Allow&Alarm=2, Deny=3, Deny&Log=4, Deny&Alarm=5.

The instances of classes that will be sent in the XML file are[39]:

```
fainPolicyRule{
              Caption: "VPN Firewalling Rule"
              Description: "..."
              CommonName: "ipvpnPolicyFirewallRule"
              PolicyKeywords[]: "CONFIGURATION"
              PolicyDecisionStrategy: "All Matching"
              PolicyRoles: ""
              Enabled: "enabled"
              ConditionListType: "CNF"
              RuleUsage: "..."
              Mandatory: TRUE
              SequencedActions: "Don't care"
              ExecutionStrategy: 1 (Do All)
              UserInfo: "NetMgr", "passwd"
}

PolicyRuleInSystem{
Antecedent: FAIN_System
Dependent: ipvpnPolicyFirewallRule
}

fainSimplePolicyCondition {
       Caption: "VPN Firewalling Condition"
       Description: "..."
       CommonName: "SourceIPCondition"
       PolicyKeywords[]: "CONFIGURATION"
```

[39] The XML mapping will be addressed as FAIN progresses and is not described here.

```
        EvaluationMethod: “MATCH”
}

PolicyConditionInPolicyRule {
        PartComponent: “SourceIPCondition”
        GroupNumber: 1
        ConditionNegated: FALSE
        GroupComponent: “ipvpnPolicyFirewallRule”
}

PolicySourceIPVariable {
        Caption: “Source IP Variable”
        Description: “...”
        CommonName: “SourceIPVariable”
        PolicyKeywords[]: ”CONFIGURATION”
        ValueTypes[]: “PolicyIPv4AddrValue”, “PolicyIPv6AddrValue”
}

PolicyVariableInSimplePolicyCondition {
        PartComponent: “SourceIPVariable”
        GroupComponent: “SourceIPCondition”
}

PolicyIPv4AddrValue {
        Caption: “IPv4 Value”
        Description: “...”
        CommonName: “IPv4AddrValue”
        PolicyKeywords[]: ”CONFIGURATION”
        IPv4AddrList[]=”147.83.106.172”
}

PolicyValueInSimplePolicyCondition {
        PartComponent: “IPv4AddrValue”
        GroupComponent: “SourceIPCondition”
}

fainSimplePolicyAction {
        Caption: “VPN Firewalling Action”
        Description: “...”
        CommonName: “ipvpnPolicyFirewallAction”
        PolicyKeywords[]: ”CONFIGURATION”
        EnforcementStrategy: “SET”
```

```
        Action: “1”
}

PolicyActionInPolicyRule {
        PartComponent: “ipvpnPolicyFirewallAction”
        GroupComponent: “ipvpnPolicyFirewallRule”
        ActionOrder: 1
}
```

7.1.3.2 FAIN Policy Domains

7.1.3.2.1 Delegation Domain

When an active node is unable to accommodate all the requests for resource allocation that it may receive, the management system may have to decide to accommodate part or all of the requested resources to a neighbouring node. What is required, is an entity that is delegated the responsibility to enforce a delegation policy to the other node. For example, an intelligent agent can be delegated the access rights from an active node, so that it can instruct another node to accept the incoming traffic.

Alternatively, it may be a requirement from the management system that a customer is delegated part of the entire management responsibility, in order to run its active service.

In either case management by delegation is necessary.

Part of the classes that can help visualise the management by delegation are:

fainCustomerIDVariable

NAME	fainCustomerIDVariable
DESCRIPTION	The ID of the Customer to whom the delegation is made
VALUE LIST	PolicyStringValue
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	CustomerCredential

fainSubjectIDVariable

NAME	fainSubjectIDVariable
DESCRIPTION	The ID of the entity that delegates authorities to the Customer
VALUE LIST	PolicyStringValue
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	fainCustomerIDVariable

fainTargetIDVariable

NAME	fainTargetIDVariable
DESCRIPTION	The ID of the target that the Customer will access
VALUE LIST	PolicyStringValue
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	fainSubjectIDVariable, fainCustomerIDVariable

fainTimePeriod

NAME	fainTimePeriod
DESCRIPTION	The range of calendar dates on which the delegation will be made
VALUE LIST	STRING
DERIVED FROM	TimePeriod

fainDelegatedResources

NAME	fainDelegatedResources
DESCRIPTION	A list of references to resources
DERIVED FROM	Resources
ABSTRACT	FALSE
PROPERTIES	fainResourceNameVariable

fainResourceNameVariable

NAME	fainResourceNameVariable
DESCRIPTION	The name of the resource
VALUE LIST	STRING
DERIVED FROM	fainDelegatedResources
ABSTRACT	FALSE
PROPERTIES	fainCustomerIDVariable, fainTargetIDVariable, fainTimePeriod, fainDelegationRestriction

fainDelegationRestriction

NAME	DelegationRestriction
DESCRIPTION	a complex type expressing access restrictions to the delegated resources (e.g. max BW = X, only event monitoring, etc)
DERIVED FROM	fainDelegatedResources
ABSTRACT	FALSE
PROPERTIES	fainTargetIDVariable, fainTimePeriod, fainResourceNameVariable

fainDelegationAction

NAME	fainDelegationAction
DESCRIPTION	Specifies the delegation parameters to be configured for a Customer.
DERIVED FROM	fainSimplePolicyAction
ABSTRACT	FALSE
PROPERTIES	fainCustomerIDVariable, fainSubjectIDVariable, fainTargetIDVariable, fainTimePeriod

fainAllowAccessAction

NAME	fainAllowAccessAction
DESCRIPTION	Allows the Customer access to the Target
DERIVED FROM	fainDelegationAction
ABSTRACT	FALSE
PROPERTIES	CustomerCredential, fainSubjectIDVariable, fainTargetIDVariable, fainTimePeriod, yes, yes+log, yes+alarm

fainDelegateResourcesAction

NAME	fainDelegateResourcesAction

DESCRIPTION	Instructs the Target which resources to grant the Customer
DERIVED FROM	fainDelegationAction
ABSTRACT	FALSE
PROPERTIES	fainCustomerIDVariable, fainTargetIDVariable, fainTimePeriod, fainResourceNameVariable

fainDisallowAccessAction

NAME	Disallow Access Action
DESCRIPTION	Disallows the Customer access to the Target
DERIVED FROM	fainDelegationAction
ABSTRACT	FALSE
PROPERTIES	CustomerCredential, fainTargetIDVariable, fainTimePeriod, yes, yes+log, yes+alarm

7.1.3.2.2 Monitoring Domain

Monitoring policies would be set in the policy-based management architecture designed in FAIN in order to obtain relevant values of the policy targets and report them to: the interested PDPs, a higher network management level, or a customer (when allowed).

The conditions that might trigger the enforcement of one monitoring policy, and therefore the report of the requested values can be either a time condition, a specific value of a property or no condition A monitoring policy with no condition, would be interpreted as a "command", the condition should be enforced immediately and only once.

The policy condition classes already defined in the FAIN Information Model (i.e. simplePolicyCondition, compoundPolicyCondition, PolicyTimePeriodCondition, PolicyVariable and PolicyValue classes and sub-classes) will fit our requirements for Monitoring policies.

The PolicyTimePeriodCondition class allows to specify time conditions on monitoring policies. Furthermore, PolicyExplicitVariable and policyValue classes are adequate to indicate a condition based on the value of a property of a class, either if it is a policy repository class or a PIB class. Only, in some cases will a new PolicyValue sub-class need to be defined.

We will have one single action class, that might be sub-classed if considered necessary. The class will have a list of properties whose values should be reported, as well as a destination property, which might optionally carry the specific PDP to which this report should be sent, and an EventId, which describes the kind of content of the properties being reported.

7.1.3.2.2.1 FAIN Monitoring Specific Classes Description

In this section a detail definition of new classes in FAIN Monitoring Information Model is given.

fainMonitoringAction

NAME	fainMonitoringAction
DESCRIPTION	This class specifies the properties to be reported and the type of properties being reported. Optionally, a specific destination can be specified as well.
DERIVED FROM	fainSimplePolicyAction
ABSTRACT	FALSE
PROPERTIES	ExplicitVariableList [ref policyExplicitVariable [0..n]] Destination EventId

The property Destination

This property is optional. When introduced, it specifies a concrete PDP to which the monitoring report should be sent. The Destination value will be a string containing the CodeID of the PDP.

NAME	Destination
DESCRIPTION	The concrete PDP to which the monitoring report should be sent.
SYNTAX	String

The property EventId

Identifies the monitoring event being sent. Monitoring events will be classified depending on the content of the properties monitored (e.g. computational resources, forwarding resources, VEs,...).

NAME	EventID
DESCRIPTION	Identifies the type of monitoring event.
SYNTAX	String

7.1.3.2.3 Service Domain

The domain of Service Policies in the FAIN policy information model has the purpose to support generic policies related with the management of services. These policies should be generic and service-independent. Policies which are entirely service-specific and require more detailed knowledge of the parameters of a service should be defined using a new service-specific policy domain, with the corresponding information model. An example of such service-specific policies are those that belong to the VPN policy domain.

The generic functions which are related to the management of a service can be split into two categories. In the first category we have the provision of computational resources to a service. The FAIN Resource Control Framework (RCF) at the node level will provide the ability to allocate both network and computational resources on a per-service granularity. Taking advantage of the RCF interface the PBANEM system will be able to assign specific CPU processing power, memory and disk space to a service. The conditions of the computational resource allocation policies will contain a service identifier. For this reason a class that identifies a specific service is required from the information model. The corresponding actions will be the allocation of a certain amount of CPU power or memory. To support these actions the information model should contain classes for the processing power and memory which are allocated to a service. An example for such a policy would be "if service = X then allocate 10% of CPU power".

The second category of service policies contains policies which are related to the management of active code. These are policies that can request the installation, removal, suspension and resumption of a particular service code module. Additionally a service may request to install a new code module, or remove an existing one.

The following classes will be needed for the FAIN Service Policy Domain

ServiceIDVariable

NAME	ServiceIDVariable
DESCRIPTION	The identifier of an active service
ALLOWED VALUE TYPES:	PolicyStringValue
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	(none)

fainServiceCPUAllocationAction

NAME	fainServiceCPUAllocationAction
DESCRIPTION	This class specifies the CPU percentage to be allocated to a single service
DERIVED FROM	fainSimplePolicyAction
ABSTRACT	FALSE
PROPERTIES	CPUPercentage

Property CPUPercentage

NAME	CPUPercentage
DESCRIPTION	The percentage of total CPU power to be allocated.
SYNTAX	Integer

fainServiceMemoryAllocationAction

NAME	fainServiceMemoryAllocationAction
DESCRIPTION	This class specifies the memory to be allocated to a service
DERIVED FROM	fainSimplePolicyAction
ABSTRACT	FALSE
PROPERTIES	MemorySize

Property MemorySize

NAME	MemorySize
DESCRIPTION	The total memory (in KB) to be allocated
SYNTAX	Integer

fainServiceStatusChangeAction

NAME	fainServiceStatusChangeAction
DESCRIPTION	This class is used to change the status of a service in an active node. It can install, suspend, resume or remove a service from an active node.
DERIVED FROM	fainSimplePolicyAction
ABSTRACT	FALSE
PROPERTIES	ServiceId

Property ServiceId

NAME	ServiceId
DESCRIPTION	An identifier for the specified service
SYNTAX	String

Property Action

NAME	Action
DESCRIPTION	The action to be performed on the specified service
SYNTAX	Integer
VALUES	Integer(ENUM) {"Install"=0;"Suspend"=1;"Resume"=2;"Remove"=3; }

fainServiceCodeUpdateAction

This action can be used in a service-specific policy, where a service can request the update of its code modules.

NAME	fainServiceCodeUpdateAction
DESCRIPTION	This action updates a service code module

DERIVED FROM		fainSimplePolicyAction
ABSTRACT	FALSE	
PROPERTIES	CodeId	
		Action

Property CodeId

NAME	CodeId
DESCRIPTION	Identifier for the code module
SYNTAX	String

Property Action

NAME		Action
DESCRIPTION		The action to be performed on the specified code module
SYNTAX		Integer
VALUES	Integer(ENUM)	
		{"Install"=0; "Remove"=1; }

7.1.3.2.3.1 Example

We present a simple example for a policy which allocates a specific percentage of CPU power to a service. “If service=multicast then allocate 20% CPU processing power “

```
fainPolicyRule{
        Caption: “Percentage of CPU”
        Description: “The policy defines the percentage of the CPU given to an application”
        CommonName: “PercentageCPUPolicyRule”
        PolicyKeywords[]: ”SERVICE”
        PolicyDecisionStrategy: “All Matching”
        PolicyRoles: “CPU consumption”
        Enabled: “enabled”
        ConditionListType: “CNF”
        RuleUsage: ”CPU reservation”
        Mandatory: TRUE
        SequencedActions: “Mandatory”
        ExecutionStrategy: 1 (Do All)
        UserInfo: “NetMgr”, “passwd”
}

PolicyRuleInSystem{
        Antecedent: FAIN_System
        Dependent: PercentageCPUPolicyRule
}

fainSimplePolicyCondition {
        Caption: “Service Condition”
        Description: “...”
        CommonName: “ServiceCondition”
        PolicyKeywords[]: ”SERVICE”
        EvaluationMethod: “MATCH”
}

PolicyConditionInPolicyRule {
        PartComponent: “ServiceCondition”
        GroupNumber: 1
        ConditionNegated: FALSE
        GroupComponent: “PercentageCPUPolicylRule”
}


PolicyServiceVariable {
```

```
        Caption: “Service Variable”
        Description: “...”
        CommonName: “ServiceVariable”
        PolicyKeywords[]: ”SERVICE”
        ValueTypes[]: “fainServiceIdVariable”,
}


PolicyVariableInSimplePolicyCondition {
        PartComponent: “ServiceVariable”
        GroupComponent: “ServiceCondition”
}

PolicyStringValue {
        Caption: “Service Id Value”
        Description: “”
        CommonName: “ServiceIdValue”
        PolicyKeywords[]: ”SERVICE”
        PolicyString=”multicast”
}


PolicyValueInSimplePolicyCondition {
        PartComponent: “ServiceIdValue”
        GroupComponent: “ServiceIdCondition”
}


fainSimplePolicyAction {
        Caption: “CPU Reservation Action”
        Description: “...”
        CommonName: “fainServiceCPUAllocationAction”
        PolicyKeywords[]: ”Service”
        EnforcementStrategy: “SET”
        CPUPercentage: “20”
}

PolicyActionInPolicyRule {
        PartComponent: “fainServiceCPUAllocationAction”
        GroupComponent: “PercentageCPUPolicyRule”
        ActionOrder: 1
}
```

7.1.3.2.4 VPN Domain

VPN policies would be set in the policy-based management architecture designed in FAIN to create, modify or remove a VPN for a user.

The conditions that might trigger the enforcement of a VPN policy, and therefore the creation, modification or removal of a VPN, can be either a time condition, a flow filter condition, a specific value of a network property or no condition..

The policy condition classes already defined in the FAIN Information Model, i.e. simplePolicyCondition, compoundPolicyCondition, PolicyTimePeriodCondition, compoundFilterCondition, PolicyVariable and PolicyValue classes and sub-classes) will fit our requirements for VPN policies both at element level, as well as at network level with some refinements. For example, different tunnels with the same IPSec parameters can be specified in the network level policy, using a compoundFilterCondition which contains a list of fainSimpleConditions all containing IPSource variables and values. Setting the property isMirrored of the CompoundFilterCondition class to true, we indicate that IPSec tunnels should be created between all these IPs in both directions.

The PolicyTimePeriodCondition class allows to specify time conditions on VPN policies. Furthermore, PolicyExplicitVariable and policyValue classes are adequate to indicate a condition based on the value of a property of a class, either a policy repository class or a PIB class. Only, in some cases will a new PolicyValue sub-class need to be defined. Finally, the compoundFilterCondition can be used to specify flow filter conditions for the creation of a VPN.

We will have two policy actions classes specific to the network level, which will be mapped into another two policy action classes specific of the element level. We will also have a policy action class which will be used at both the network level and the element level. The actions defined are related to three VPN related domains, security based on IPSec, paths with certain QoS based on MPLS, and a firewalling action.

All action classes are based on the VPN Information model draft proposed by the IETF with small modifications in order to adapt them to our specific needs. These action classes will be detailed described in the section below, as well as the mapping between the network level policies and the element level policies when needed.

7.1.3.2.4.1 FAIN VPN Specific Classes Description

In this section a detailed definition of new classes in FAIN VPN Information Model is given, as well as issues related with the mapping between the network and element level policies where appropriate.

7.1.3.2.4.1.1 IPSec

Network Level: fainVPNIPSecAction

NAME	fainIPSecAction
DESCRIPTION	This class specifies the properties of the IPSec tunnel or tunnels requested.
DERIVED FROM	fainSimplePolicyAction
ABSTRACT	FALSE
PROPERTIES	IsEncrypted
	IsAuthenticated
	IPSecAuthPref
	IPSecEncPref
	IPDHGroupPref
	IKEAuthPref
	IKEEncPref
	IKEDHGroupPref
	IKEPeerAuthMethodPref

The property IsEncrypted

This property is an enumerated type with four possible values: 0, 1, 2, 3. The “0” means no encryption; “1” encryption in the specified direction; “2” encryption in the mirrored direction; and “3” encryption in both directions.

NAME	IsEncrypted
DESCRIPTION	Indicates whether encryption should be applied in the tunnel.
SYNTAX	Integer

The property IsAuthenticated

This property is an enumerated type with four possible values: 0, 1, 2, 3. The “0” means no authentication; “1” authentication in the specified direction; “2” authentication in the mirrored direction; and “3” authentication in both directions.

NAME	IsAuthenticated
DESCRIPTION	Indicates whether authentication should be applied in the tunnel.
SYNTAX	Integer

The property IPSecAuthPref

The value of this property, if present, specifies the IPSec authentication algorithm preferred by the User.

NAME	IPSecAuthPref
DESCRIPTION	Indicates the preferred IPSec Authentication algorithm
SYNTAX	String

The property IPSecEncPref

The value of this property, if present, specifies the IPSec encryption algorithm preferred by the User.

NAME	IPSecAuthPref
DESCRIPTION	Indicates the preferred IPSec encryption algorithm
SYNTAX	String

The property IPSecDHGroupPref

The value of this property, if present, specifies the IPSec Diffie-Hellman group preferred by the User.

NAME	IPSecDHGroupPref
DESCRIPTION	Indicates the preferred IPSec Diffie-Hellman group.
SYNTAX	String

The property IKEAuthPref

The value of this property, if present, specifies the IKE authentication algorithm preferred by the User.

NAME	IKEAuthPref
DESCRIPTION	Indicates the preferred IKE Authentication algorithm
SYNTAX	String

The property IKEEncPref

The value of this property, if present, specifies the IKE encryption algorithm preferred by the User.

NAME	IKEAuthPref
DESCRIPTION	indicates the preferred IKE encryption algorithm
SYNTAX	String

The property IKEDHGroupPref

The value of this property, if present, specifies the IKE Diffie-Hellman group preferred by the User.

NAME	IKEDHGroupPref
DESCRIPTION	indicates the preferred IKE Diffie-Hellman group.
SYNTAX	String

The property IKEPeerAuthMethodPref

This property is an enumerated type with six possible values, that shows the IKE peer authentication method preferred by the User :

- 1: Pre-shared key
- 2: DSS signatures
- 3: RSA signatures

- 4: Encryption with RSA
- 5: Revised encryption with RSA
- 6: Kerberos

NAME	IKEPeerAuthMethodPref
DESCRIPTION	Indicates the preferred IKE peer authentication method.
SYNTAX	Integer

Element Level: fainIPSecAction

NAME fainipvpnEncryptionAction
DESCRIPTION This class specifies the necessary properties for the creation of an IPSec tunnel.
DERIVED FROM fainSimplePolicyAction
ABSTRACT FALSE
PROPERTIES VPNId
IkeAuthentication
IkeEncryption
IkeDHGroup
IkeTimeout
IkeTrafficBasedExpiry
IPSecAuthentication
IPSecEncryption
IPSecDHGroup
IPSecTimeout
IPSecTrafficBasedExpiry
IkePeerAuthenticationMethod

VPNId

The VPNId property value indicates to the element level to which VPN this tunnel has to be created.

NAME	VPNId
DESCRIPTION	Indicates the VPN that contains the requested tunnel.
SYNTAX	String

The property IkeAuthentication

The property specifies the authentication algorithm to be used.

NAME IkeAuthentication
DESCRIPTION The property that specifies the authentication algorithm
SYNTAX String

The property IkeEncryption

The property specifies the encryption algorithm to be used.

NAME IkeEncryption
DESCRIPTION The property that specifies the encryption algorithm
SYNTAX String

The property IkeDHGroup

The property specifies the DHGroup to be used during IKE negotiations

NAME IkeDHGroup

DESCRIPTION The property that specifies the DHGroup to be used during IKE negotiations

SYNTAX String

The property IkeTimeout

The property specifies the IKE Timeout to be used.

NAME IkeTimeout

DESCRIPTION The property that specifies the IKE timeout

SYNTAX Integer

The property IkeTrafficBasedExpiry

The property specifies the IKE Traffic based expiry to be used.

NAME IkeTrafficBasedExpiry

DESCRIPTION The property that specifies the IKE traffic based expiry to be used

SYNTAX Integer

The property IPSECAuthentication

The property specifies the authentication algorithm to be used.

NAME IPSECAuthentication

DESCRIPTION The property that specifies the authentication algorithm

SYNTAX String

The property IPSECEncryption

The property specifies the encryption algorithm to be used.

NAME IPSECEncryption

DESCRIPTION The property that specifies the encryption algorithm

SYNTAX String

The property IPSECDHGroup

The property specifies the DHGroup to be used during IPSEC negotiations

NAME IPSECDHGroup

DESCRIPTION The property that specifies the DHGroup to be used during the Phase II negotiations

SYNTAX String

The property IPSECTimeout

The property specifies the IPSEC Key Timeout to be used.

NAME IPSECTimeout

DESCRIPTION The property that specifies the IPSEC Key timeout

SYNTAX Integer

The property IPSECTrafficBasedExpiry

The property specifies the IPSEC Traffic based Key expiry to be used.

NAME IPSECTrafficBasedExpiry

DESCRIPTION The property that specifies the IPSEC traffic based Key expiry to be used

SYNTAX Integer

The property IkePeerAuthenticationMethod

The method used by the Ike peers to authenticate each other. It is an enumerated value with seven possible values:

· 0 - a special value which indicates that this particular proposal should be repeated once for each authentication method that corresponds to the credentials installed on the machine. For example, if the system has a pre-shared key and a certificate, a proposal list could be constructed which includes a proposal that specifies pre-shared key and proposals for any of the public-key authentication methods.

· 1 - Pre-shared key

· 2 - DSS signatures

· 3 - RSA signatures

· 4 - Encryption with RSA

· 5 - Revised encryption with RSA

· 6 - Kerberos (issue with assigning number)

NAME	IkePeerAuthenticationMethod
DESCRIPTION	The property that specifies the method used by the Ike peers to authenticate each other
SYNTAX	integer

7.1.3.2.4.1.2 Mapping between network and element level IPSec policies

Since one network level IPSec policy can specify several IPSec tunnels with the same characteristics. This network level policy has to be translated to one element level policy for each tunnel that should be sent to the appropriate PBANEM node to be enforced correctly. This will imply a mapping of the policy conditions and policy actions.

The policy condition mapping is straightforward if the network level policy specifies just one IPSec tunnel. When the network level policy specifies several IPSec tunnels, the mapping of the policy condition should be made by making one element level policy, with the correspondent source and destination IP address pair, for each IPSec tunnel requested.

The mapping of the network level policy action to the element level IPSec policy action does not depend on the number of tunnels requested at the network level policy, just on the property values.

The network level has to add to the element level policy action the property VPNId that indicates the VPN to which that IPSec tunnel has to be added. Moreover, the User can introduce a series of requests in the network level policy for its IPSec tunnels. It can determine whether it wants encryption, authentication or both, in the tunnel, or the preferred algorithms to be used at the different levels of the IPSec tunnel creation.

If the network level policy action indicates that either encryption or authentication is not desired, the element level policy action properties IPSecAuthentication or IPSecEncryption would be set to the values NoAuth or NoEnc respectively.

If the User does not specify a preferred algorithm, the network level can choose the algorithms to be used in the IPSec tunnel creation based on the algorithms supported by its routers. Furthermore, if the network level policy specifies several IPSec tunnels, with no preferred algorithm, each tunnel can be created with different algorithms depending on the algorithms supported by the routers implied in the creation of each tunnel.

On the other hand, if the User expresses in the network level policy their desire to use some concrete algorithms, the network level has to check whether the involve routers support these algorithms or whether they can be extended to support them. Based on this information, the network level has to take a decision. If extension of the algorithms supported by the router is needed, the network level management system has to request the installation of these algorithms in the routers through the correspondent interactions with the ASP system. Finally, if one concrete algorithm is requested by the user and accepted by the network level, it would be mapped to element level policies introducing the algorithm to be used in the appropriate element level policy action property.

Element level timeout and traffic based expiry property values are set by the element level according to its necessities and preferences.

7.1.3.2.4.2 FAIN MPLS Specific Classes Description

Network Level: fainVPNMPLSAction

NAME	fainVPNMPLSAction
DESCRIPTION	This class specifies the properties of MPLS LSPs requested.
DERIVED FROM	fainSimplePolicyAction
ABSTRACT	FALSE
PROPERTIES	ResourceClassAffinity
	QoSClass
	RequestedBW
	ActionMode

The property ResourceClassAffinity

This property allows the user to indicate a special kind of resource class that he explicitly desires or rejects to be used in the creation of the LSP. A User may specify that it desires the LSP to be created over OC-48 links.

The syntax of the property is a string of the form:

<resource-class, affinity>; <resource-class, affinity>; ..

where affinity is a boolean that when set to 1 indicates desired, 0 indicates rejected.

NAME	ResourceClassAffinity
DESCRIPTION	Indicates concrete resource classes desired or rejected.
SYNTAX	String

The property QoSClass

This property is an enumerated type with three possible values: 0, 1, 2. The different values will indicate a different class of QoS of the LSP requested (i.e. (2)gold, (1)siver or (0)bronze).

NAME	QoSClass
DESCRIPTION	Indicates the QoS class.
SYNTAX	Integer

The property RequestedBW

Indicates the token rate desired for this LSP.

NAME	RequestedBW
DESCRIPTION	Indicates the token rate desired for this LSP.
SYNTAX	Integer

The property ActionMode

This property is an enumerated type with three possible values: 0, 1, 2. The different values will indicate the action to be applied in the LSP requested (i.e. (0)create, (1)remove or (2)update).

NAME	ActionMode
DESCRIPTION	Indicates the action to be applied on the LSP.
SYNTAX	Integer

Element Level: fainipvpnPolicyTrafficTrunkAction

NAME fainipvpnPolicyTrafficTrunkAction

DESCRIPTION This class specifies the properties of an MPLS LSP requested.

DERIVED FROM fainSimplePolicyAction

ABSTRACT FALSE

PROPERTIES VPNId

ipvpnTrafficTrunk [ref mplsPolicyTrafficTrunk[0..n]]

ipvpnRouteSpecification[ref mplsPolicyRouteSpec [0..n]]

ipvpnTrafficProfile [ref qosPolicyPRTrfcProf[0..n]]

ipvpnFEC [ref mplsPolicyFEC [0..1]]

ActionMode

The property VPNId

The VPNId property value indicates to the element level to which VPN the LSP pertains.

NAME	VPNId
DESCRIPTION	Indicates the VPN that contains the requested LSP.
SYNTAX	String

The property ipvpnTrafficTrunk

This property is a reference to the mplsPolicyTrafficTrunk class (reference to classes is the way usd by CIM to express complex types in properties).

The mplsPolicyTrafficTrunk class and its properties are described below.

mplsPolicyTrafficTrunk

NAME mplsPolicyTrafficTrunk

DESCRIPTION A class with several properties for describing an MPLS traffic trunk.

DERIVED FROM LogicalElement

ABSTRACT False

PROPERTIES mpResourceClassAffinity,

mpPreemption,

mpPriority,

mpResilience,

mpTrafficProportion,

mpReoptimizationFreq,

Roles

The property mpResourceClassAffinity

This property allows the user to indicate a special kind of resource class that they explicitly desire or wish to reject being used in the creation of the LSP. A User may specify that they desires the LSP to be created over OC-48 links.

The syntax of the property is a string of the form:

<resource-class, affinity>; <resource-class, affinity>; ..

where affinity is a boolean that when set to 1 indicates desired, 0 indicates rejected.

NAME	mpResourceClassAffinity
DESCRIPTION	Indicates concrete resource classes desired or rejected.
SYNTAX	String

The property mpPreemption

The preemption attribute (see [82]) determines whether a traffic trunk can preempt another traffic trunk from a given path, and whether it can be preempted by another traffic trunk. Preemption can be used to assure that high priority traffic trunks can always be routed through relatively favourable paths within a differentiated services environment. Preemption can also be used to implement various prioritized restoration policies following fault events.

The preemption property can take one of four values, with the following semantics:

1. preemptor-enabled: can preempt lower priority preemptable

   traffic trunks
2. non-preemptor: cannot preempt other traffic trunks
3. preemptable: can be preempted by higher priority preemptor-

   enabled traffic trunks
4. non-preemptable: cannot be preempted.

It is trivial to see that some of the preemptive modes are mutually exclusive. Using the numbering scheme depicted above, the feasible preemptive mode combinations for a given traffic trunk are as follows:

(1, 3), (1, 4), (2, 3), and (2, 4). The (2, 4) combination should be the default.

A traffic trunk, say "A", can preempt another traffic trunk, say "B", only if *all* of the following five conditions hold:

1. "A" has a relatively higher priority than "B"
2. "A" contends for a resource utilized by "B"
3. The resource cannot concurrently accommodate "A" and "B"

   based on certain decision criteria
4. "A" is preemptor enabled
5. "B" is preemptable.

NAME	mpPreemption
DESCRIPTION	Contains preemptability information
SYNTAX	Integer (MUST be in the range 1-4)

The property mpPriority

The priority of a traffic trunk is described by this property. The priority property defines the relative importance of traffic trunks. If a constraint-based routing framework is used with MPLS, priorities can be used to determine the order in which path selection is done for traffic trunks at connection establishment and under fault scenarios. Priorities are also important in implementations permitting preemption because they can be used to impose a partial order on the set of traffic trunks according to which preemptive policies can be applied. The priority of a traffic trunk, along with its preemptability information (see the description of the mpPreemption property in the previous section), determines when it will preempt and/or be preempted by other traffic trunks.

NAME mpPriority

DESCRIPTION Priority for this traffic trunk.

SYNTAX Integer

The property mpResilience

The mpResilience property indicates the recovery procedure to be applied to traffic trunks whose paths are impacted by faults. More specifically, it contains a boolean value that determines whether the target traffic trunk is to be rerouted or not when segments of its path fail.

NAME mpResilience

DESCRIPTION If set to true, this traffic trunk should be rerouted in case of failure.

SYNTAX boolean

The property mpTrafficProportion

This property is used to indicate the relative proportion of traffic to be carried by parallel traffic trunks. This enables one to perform load distribution across multiple parallel traffic trunks between two nodes. In many practical situations, the aggregate traffic between two nodes may be such that no single link can carry the load. In this case, the only feasible solution is to appropriately divide the aggregate traffic into sub-streams and route the sub-streams through multiple paths between the two nodes. This problem can be addressed by instantiating multiple traffic trunks between the two nodes, such that each traffic trunk carries a proportion of the aggregate traffic. The proportion of traffic carried by each such trunk is specified by the mpTrafficProportion property.

NAME mpTrafficProportion

DESCRIPTION Proportion of traffic to be carried by this traffic trunk, specified as a percentage from 0 to 100.

SYNTAX Integer

The property mpReoptimizationFreq

Due to changes in network and traffic characteristics, there may be a need to periodically change the paths of traffic trunks for optimization purposes. This should not be done too frequently as this could adversely affect the stability of the network. This property indicates how often such reoptimization should be performed.

NAME mpReoptimizationFreq

DESCRIPTION Indicates how frequently reoptimization should be performed for this traffic trunk. If the value of this property is set to zero, this indicates that reoptimization should not be performed.

SYNTAX Integer

The property Roles

The Roles property specifies the set of roles this TT may have. This property is defined in the CIM Core Information model and therefore its definition is not repeated here. See PCIM [62] for an explanation of how roles are used.

The property ipvpnRouteSpecification

This property is a reference to the mplsPolicyRouteSpec class (reference to classes is the way usd by CIM to express complex types in properties).

The mplsPolicyRouteSpec class and its properties are described below.

fainmplsPolicyRouteSpec

NAME fainmplsPolicyRouteSpec

DESCRIPTION A class describing an MPLS route specification.

DERIVED FROM mplsPolicyRouteSpec

ABSTRACT False

PROPERTIES mpIngressIPAddress,
MpEgressIPAddress,
interMandatoryNodes

The property mpIngressIPAddress

Ingress IP address for this MPLS route.

NAME mpIngressIPAddress

DESCRIPTION Ingress IP address for this MPLS route.

SYNTAX string

The property mpEgressIPAddress

Egress IP address for this MPLS route.

NAME mpEgressIPAddress

DESCRIPTION Egress IP address for this MPLS route.

SYNTAX string

The property interMandatoryNodes

IP address of mandatory nodes where the LSP should be created upon. This property has been add to the mplsPolicyRouteSpec class in order to allow the network level to indicate a concrete route for the LSP.

NAME interMandatoryNodes

DESCRIPTION Intermediate mandatory hops.

SYNTAX string

The property ipvpnTrafficProfile

This property is a reference to the qosPolicyPRTrfcProf class (reference to classes is the way usd by CIM to express complex types in properties).

The qosPolicyPRTrfcProf class and its properties are described below.

qosPolicyPRTrfcProf

NAME qosPolicyPRTrfcProf

DERIVED FROM gpsPolicyTrfcProf

ABSTRACT False

PROPERTIES qpPRRate,
qpPRNormalBurst,
qpPRExcessBurst

The Property qpPRRate

This is a non-negative integer that defines the token rate in kilobits per second. A rate of zero means that all packets will be out of profile. The attribute is defined as follows:

NAME qpPRRate
SYNTAX Integer (must be non-negative)

The Property qpPRNormalBurst

This attribute is an integer that defines the normal size of a burst measured in bytes. The attribute is defined as follows:

NAME qpPRNormalBurst
SYNTAX Integer (must be non-negative)

The Property qpPRExcessBurst

This attribute is an integer that defines the excess size of a burst measured in bytes. The attribute is defined as follows:

NAME qpPRExcessBurst
SYNTAX Integer (must be non-negative)

The property ipvpnFEC

This property is a reference to the mplsPolicyFEC class (reference to classes is the way used by CIM to express complex types in properties).

The mplsPolicyFEC class and its properties are described below.

mplsPolicyFEC

The mplsPolicyFEC specifies the Forwarding Equivalence Class of an LSP. The FECs may differ depending on the application of MPLS. This class does not have any property. The filter is specified through the mplsFECFilterSet association that associates a FilterList [77]with this class.

NAME mplsPolicyFEC

DESCRIPTION A Forwarding Equivalence Class of a Traffic Trunk of an LSP

DERIVED FROM Policy

ABSTRACT FALSE

PROPERTIES

The property ActionMode

This property is an enumerated type with three possible values: 0, 1, 2. The different values will indicate the action to be applied in the LSP requested (i.e. (0) create, (1) remove or (2) update).

NAME	ActionMode
DESCRIPTION	Indicates the action to be applied on the LSP.
SYNTAX	Integer

7.1.3.2.4.2.1 Mapping between network and element level MPLS policies.

The mapping of MPLS policy conditions at the network level and the element level follows the same guidelines as the IPSec policy conditions.

The mapping of the MPLS policy actions is quite simple as well. The mpResourceClassAffinity property can be mapped directly from the network level ResourceClassAffinity property.

The network level QoSClass property controls the value of some concrete element level properties that deal with the QoS of the LSP created. The value of these element level properties has to be set by the network management level according to the network operator policy and knowledge. These values are: mpPreemption, mpPriority, mpResilience, mpReoptimizationFreq, qpPRNormalBurst and qpPRExcessBurst. The better QoS class chosen, the better values these attributes will have. For example, with GOLD QoS class, the LSP created for the User will be able to preempt other traffic trunks, will not be preemptable by other trunks, will have a high priority, will be recoverable if a fault occurs (mpResilience), it could be optimised periodically, and finally it will have better burst and excess-burst parameters.

The mpTrafficProportion property will be set by the network level according to the topology and network resource information it has.

The RequestedBW parameter will be mapped directly with the qpRate property of the qosPolicyPRTrfcProf class. The ActionMode property at the network level is directly mappedto the ActionMode property of the element level.

The policy condition compoundFilterCondition at the network level, along with the topology and resource information available at the network management system, will be used to set the values of the mplsPolicyFEC, mpIngressIPAddress and mpEgressIPAddress at the element level. The network management system can also specify a concrete route for the LSP to be created. In such a case, it will use the InterMandatoryNodes property to specify that route.

7.1.3.2.4.3 Firewall

Network and Element level: fainipvpnPolicyFirewallAction

This action is applicable at both network and element level. The only difference is that the policy condition can specify different flows to which the same firewall action can be applied. While at the element level, one element level policy should be created for each router that has to act as a firewall and apply these firewall actions.

NAME	fainipvpnPolicyFirewallAction
DESCRIPTION	This class specifies a firewall to be applied to a VPN.
DERIVED FROM	fainSimplePolicyAction
ABSTRACT	FALSE
PROPERTIES	VPNId
	Action

The property Action

The action defines the type of firewall action to be enforced. It is an enumerated property with six possible values:

- "Allow"=0
- “Allow&Log"=1
- "Allow&Alarm"=2
- "Deny"=3
- "Deny&Log"=4
- "Deny&Alarm" =5

NAME	Action
DESCRIPTION	The firewall action to be enforced
SYNTAX	Integer

7.1.3.2.5 QoS Domain

A policy domain is designed to support the QoS Mgmt. Delegation scenario using the multi-PDP PBANEM architecture. It is derived from the core FAIN information model by adding class definitions which are scenario- specific.

The scenario makes use of active technology and tries to manage the resource allocation on a per-customer basis. The overhead of the management is delegated to individual customers to fulfill individual users' bandwidth needs. Policy system modules (e.g. a QoS PDP) can be dynamically provisioned via interfaces with the service provisioning framework. It aims to demonstrate the advantages of active networking to ease the network bandwidth management procedure.

In this scenario, a policy domain is supposed to support the following functions:

- QoS provisioning: Intserv and Diffserv is supported, therefore the QoS policy information model [83] is considered a standard model and will be reused and extended, if necessary.
- Delegation: a network is partitioned into virtual networks, management of virtual resources are delegated to customers according to SLAs. Policies are defined and transferred from network managers to network elements, and perform partitioning and set-up delegation parameters, e.g. customer credential for authentication.
- Provision: the policy system modules will be dynamically downloaded and updated, if possible. Policies need to specify how this can be done for those QoS modules such as an Diffserv PDP, its traffic conditioning blocks, metering block, etc.

In the following, classes are defined according to these function categories.

7.1.3.2.5.1 QoS Policy

The class hierarchy of QoS policies within FAIN policy core information model is depicted as below. The proposed extensions are highlighted.

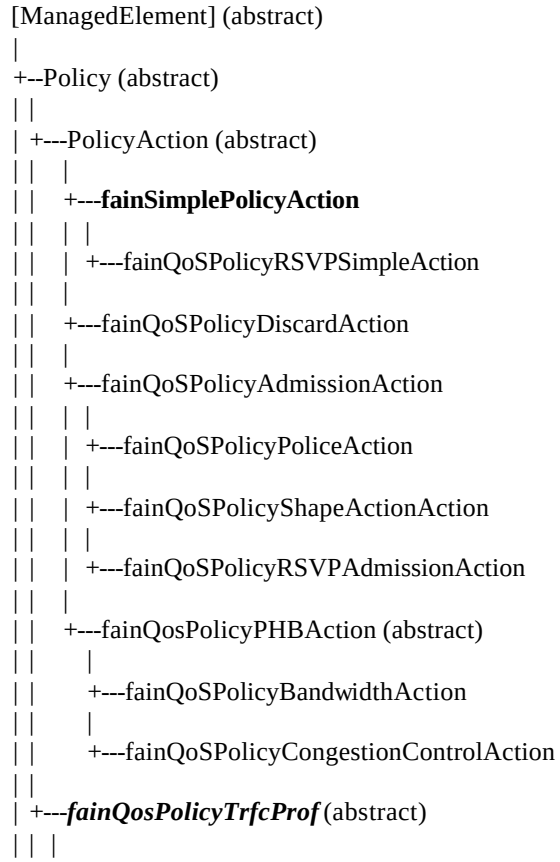

Figure 66 - QoS Policy Class Hierarchy

For a more detailed description of each class and its properties, refer to [83].

7.1.3.2.5.2 Delegation Policy

fainNetPartitionAction

NAME	fainNetPartitionAction
DESCRIPTION	This class specifies the partitioned network configuration to be allocated to a customer
DERIVED FROM	fainSimplePolicyAction

ABSTRACT FALSE

PROPERTIES topologyVirtualNet, bandwidthVirtualNet

fainElementPartitionAction

NAME fainElementPartitionAction

DESCRIPTION This class specifies the partitioned element configuration to be allocated to a customer

DERIVED FROM fainSimplePolicyAction

ABSTRACT FALSE

PROPERTIES idVirtualNet, idInterface, idOutputQuque, bandwidthPercentage, priority

fainDelegationAction

NAME fainDelegationAction

DESCRIPTION This class specifies the delegation parameters to be configured for a customer

DERIVED FROM fainSimplePolicyAction

ABSTRACT FALSE

PROPERTIES customerCredential, accessControlEnabled

fainCustomerIDVariable

NAME fainCustomerIDVariable

DESCRIPTION The identifier of a customer.

ALLOWED VALUE TYPES: PolicyStringValue

DERIVED FROM PolicyImplicitVariable

ABSTRACT FALSE

PROPERTIES credential, certificate

7.1.3.2.5.3 Provision Policy

fainQoSPDPProvisionAction

NAME fainQoSPDPProvisionAction

DESCRIPTION This class specifies the additional parameters for deploying, updating and reconfiguring QoS policy decision modules realizing QoS management

DERIVED FROM fainServiceCodeUpdateAction

ABSTRACT FALSE

PROPERTIES customerCredential, idPEPgroup, idMonitor

fainQoSPEPProvisionAction

NAME fainQoSPEPProvisionAction

DESCRIPTION This class specifies the additional parameters for deploying, updating and reconfiguring QoS policy enforcement modules realizing QoS configuration

DERIVED FROM fainServiceCodeUpdateAction

ABSTRACT FALSE

PROPERTIES customerCredential, idPDP, idMonitor

8 APPENDIX C: XML POLICY MAPPING - CODE EXAMPLE

In this appendix, in order to ease the comprehension of how the XML mapping has been done we provide two examples. The examples are one XML-Schema mapping which will validate a XML fainPolicyRule.

The rule is given here as an example, is taken from the VPN information model proposed by the IETF [31], as the example given in chapter 2.4. However, it is slightly more complex than that of chapter 2.4. The condition of the policy is a CompoundPolicyCondition formed with three fainSimplePolicyConditions (i.e. flow direction, IP source and IP destination). The action has 11 properties related with the IPSec protocol.

8.1 XML-SCHEMA EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.ist-fain.org/Schema" xmlns="http://www.ist-fain.org/Schema"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
    <xsd:notation name="FAIN-IPSec" public="http://www.ist-fain.org/schemas/FAIN/IPSec"/>
    <!-- contents of XML Schema document goes here -->
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
    IPSec in XML Schema example.
    </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="fainPolicyRule" type="fainPolicyRuleType"/>
    <xsd:element name="CommonElements">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Caption" type="xsd:string"/>
                <xsd:element name="Description" type="xsd:string"/>
                <xsd:element name="CommonName" type="xsd:string"/>
                <xsd:element name="PolicyKeywords">
                    <xsd:simpleType>
                        <xsd:list itemType="PolicyKeywordValue"/>
                    </xsd:simpleType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PolicyActionKeys">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="CreationClassName">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="PolicyActionName">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PolicyConditionKeys">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="CreationClassName">
```

```
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="PolicyConditionName">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PolicyVariableKeys">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="CreationClassName">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="PolicyVariableName">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PolicyValueKeys">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="CreationClassName">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="PolicyValueName">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="fainPolicyRuleType">
        <xsd:sequence>
```

```
<xsd:element ref="CommonElements" minOccurs="0"/>
<xsd:element name="CreationClassName">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="PolicyRuleName">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="PolicyDecisionStrategy">
    <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
            <xsd:enumeration value="1"/>
            <xsd:enumeration value="2"/>
            <xsd:enumeration value="3"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="PolicyRoles">
    <xsd:simpleType>
        <xsd:list itemType="xsd:string"/>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="Enabled">
    <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
            <xsd:enumeration value="1"/>
            <xsd:enumeration value="2"/>
            <xsd:enumeration value="3"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="ConditionListType">
    <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
            <xsd:enumeration value="1"/>
            <xsd:enumeration value="2"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="RuleUsage" type="xsd:string"/>
<xsd:element name="Mandatory" type="xsd:boolean"/>
<xsd:element name="SequencedActions">
    <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
            <xsd:enumeration value="1"/>
            <xsd:enumeration value="2"/>
            <xsd:enumeration value="3"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="ExecutionStrategy">
    <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
            <xsd:enumeration value="1"/>
            <xsd:enumeration value="2"/>
            <xsd:enumeration value="3"/>
            <xsd:enumeration value="4"/>
        </xsd:restriction>
    </xsd:simpleType>
```

```
            </xsd:element>
            <xsd:element name="UserInfo">
                <xsd:simpleType>
                    <xsd:list itemType="xsd:string"/>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element name="ConditionReference" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="PolicyCondition" type="PolicyConditionType"/>
                    </xsd:sequence>
                    <xsd:attribute name="GroupNumber" type="xsd:decimal"/>
                    <xsd:attribute name="ConditionNegated" type="xsd:boolean"/>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="ActionReference" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="PolicyAction" type="PolicyActionType"/>
                    </xsd:sequence>
                    <xsd:attribute name="ActionOrder" type="xsd:decimal"/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="PolicyConditionType" abstract="true">
        <xsd:sequence>
            <xsd:element ref="CommonElements" minOccurs="0"/>
            <xsd:element ref="PolicyConditionKeys"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CompoundFilterConditionType">
        <xsd:complexContent>
            <xsd:extension base="PolicyConditionType">
                <xsd:sequence>
                    <xsd:element name="IsMirrored" type="xsd:boolean"/>
                    <xsd:element name="ConditionReference" minOccurs="0" maxOccurs="unbounded">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="PolicyCondition" type="PolicyConditionType"/>
                            </xsd:sequence>
                            <xsd:attribute name="GroupNumber" type="xsd:decimal"/>
                            <xsd:attribute name="ConditionNegated" type="xsd:boolean"/>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="fainSimplePolicyConditionType">
        <xsd:complexContent>
            <xsd:extension base="PolicyConditionType">
                <xsd:sequence>
                    <xsd:element name="EvaluationMethod" type="xsd:string"/>
                    <xsd:element name="PolicyVariable" type="PolicyVariableType" minOccurs="0"
maxOccurs="unbounded"/>
                    <xsd:element name="PolicyValue" type="PolicyValueType" minOccurs="0"
maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="PolicyVariableType">
        <xsd:sequence>
            <xsd:element ref="CommonElements" minOccurs="0"/>
            <xsd:element ref="PolicyVariableKeys"/>
            <xsd:element name="ValueTypes">
                <xsd:simpleType>
                    <xsd:list itemType="xsd:string"/>
                </xsd:simpleType>
```

```
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="PolicyValueType">
        <xsd:sequence>
            <xsd:element ref="CommonElements" minOccurs="0"/>
            <xsd:element ref="PolicyValueKeys"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="PolicyStringValueType">
        <xsd:complexContent>
            <xsd:extension base="PolicyValueType">
                <xsd:sequence>
                    <xsd:element name="StringList">
                        <xsd:simpleType>
                            <xsd:list itemType="xsd:string"/>
                        </xsd:simpleType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="PolicyIPv4ValueType">
        <xsd:complexContent>
            <xsd:extension base="PolicyValueType">
                <xsd:sequence>
                    <xsd:element name="IPv4AddrList">
                        <xsd:simpleType>
                            <xsd:list itemType="IPv4Addr"/>
                        </xsd:simpleType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="IPv4Addr">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\[0-255].[0-255].[0-255].[0-255]/[0-32]"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="PolicyActionType" abstract="true">
        <xsd:sequence>
            <xsd:element ref="CommonElements" minOccurs="0"/>
            <xsd:element ref="PolicyActionKeys"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CompoundActionType">
        <xsd:complexContent>
            <xsd:extension base="PolicyActionType">
                <xsd:sequence>
                    <xsd:element name="SequencedActions">
                        <xsd:simpleType>
                            <xsd:restriction base="xsd:decimal">
                                <xsd:enumeration value="1"/>
                                <xsd:enumeration value="2"/>
                                <xsd:enumeration value="3"/>
                            </xsd:restriction>
                        </xsd:simpleType>
                    </xsd:element>
                    <xsd:element name="ExecutionStrategy">
                        <xsd:simpleType>
                            <xsd:restriction base="xsd:decimal">
                                <xsd:enumeration value="1"/>
                                <xsd:enumeration value="2"/>
                                <xsd:enumeration value="3"/>
                                <xsd:enumeration value="4"/>
                            </xsd:restriction>
                        </xsd:simpleType>
                    </xsd:element>
                    <xsd:element name="ActionReference" minOccurs="0" maxOccurs="unbounded">
```

```
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="PolicyAction" type="PolicyActionType"/>
                            </xsd:sequence>
                            <xsd:attribute name="ActionOrder" type="xsd:decimal"/>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="fainSimplePolicyActionType" abstract="true">
        <xsd:complexContent>
            <xsd:extension base="PolicyActionType">
                <xsd:sequence>
                    <xsd:element name="EnforcementStrategy" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="EncActionType">
        <xsd:complexContent>
            <xsd:extension base="fainSimplePolicyActionType">
                <xsd:sequence>
                    <xsd:element name="IkeAuthentication" type="xsd:string"/>
                    <xsd:element name="IkeEncryption" type="xsd:string"/>
                    <xsd:element name="IkeDHGroup" type="xsd:string"/>
                    <xsd:element name="IkeTimeout" type="xsd:integer"/>
                    <xsd:element name="IkeTrafficBasedExpiry" type="xsd:integer"/>
                    <xsd:element name="IPSecAuthentication" type="xsd:string"/>
                    <xsd:element name="IPSecEncryption" type="xsd:string"/>
                    <xsd:element name="IPSecDHGroup" type="xsd:string"/>
                    <xsd:element name="IPSecTimeout" type="xsd:integer"/>
                    <xsd:element name="IPSecTrafficBasedExpiry" type="xsd:integer"/>
                    <xsd:element name="IkePeerAuthenticationMethod">
                        <xsd:simpleType>
                            <xsd:restriction base="xsd:short">
                                <xsd:enumeration value="1"/>
                                <xsd:enumeration value="2"/>
                                <xsd:enumeration value="3"/>
                                <xsd:enumeration value="4"/>
                                <xsd:enumeration value="5"/>
                                <xsd:enumeration value="6"/>
                            </xsd:restriction>
                        </xsd:simpleType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="PolicyKeywordValue">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="UNKNOWN"/>
            <xsd:enumeration value="CONFIGURATION"/>
            <xsd:enumeration value="USAGE"/>
            <xsd:enumeration value="SECURITY"/>
            <xsd:enumeration value="SERVICE"/>
            <xsd:enumeration value="MOTIVATIONAL"/>
            <xsd:enumeration value="INSTALLATION"/>
            <xsd:enumeration value="EVENT"/>
            <xsd:enumeration value="DELEGATION"/>
            <xsd:enumeration value="FAULT"/>
            <xsd:enumeration value="MONITORING"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

8.2 XML POLICY INSTANCE MAPPING

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT beta 4 build Jan 12 2001 (http://www.xmlspy.com) by Alexander Falk (Altova, Inc.) -->
<a:fainPolicyRule xmlns:a="http://www.ist-fain.org/Schema" xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation="http://www.ist-fain.org/Schema
IPSec_in_XML.xsd">
    <CreationClassName Key="true">fainPolicyRule</CreationClassName>
    <PolicyRuleName Key="true">IPSecPolicy</PolicyRuleName>
    <PolicyDecisionStrategy>2</PolicyDecisionStrategy>
    <PolicyRoles>VPN</PolicyRoles>
    <Enabled>1</Enabled>
    <ConditionListType>2</ConditionListType>
    <RuleUsage>VPNCreation</RuleUsage>
    <Mandatory>true</Mandatory>
    <SequencedActions>3</SequencedActions>
    <ExecutionStrategy>3</ExecutionStrategy>
    <UserInfo>Admin admpass</UserInfo>
    <ConditionReference GroupNumber="1" ConditionNegated="false">
        <PolicyCondition xsi:type="a:CompoundFilterConditionType">
            <a:PolicyConditionKeys>
                <CreationClassName Key="true">CompoundFilterCondition</CreationClassName>
                <PolicyConditionName Key="true">VPNFilterCondition</PolicyConditionName>
            </a:PolicyConditionKeys>
            <IsMirrored>false</IsMirrored>
            <ConditionReference GroupNumber="1" ConditionNegated="false">
                <PolicyCondition xsi:type="a:fainSimplePolicyConditionType">
                    <a:PolicyConditionKeys>
                        <CreationClassName Key="true">fainSimplePolicyCondition</CreationClassName>
                        <PolicyConditionName Key="true">FlowDirectionCondition</PolicyConditionName>
                    </a:PolicyConditionKeys>
                    <EvaluationMethod>Match</EvaluationMethod>
                    <PolicyVariable>
                        <a:PolicyVariableKeys>
                            <CreationClassName Key="true">PolicyFlowDirectionVariable</CreationClassName>
                            <PolicyVariableName Key="true">FlowVariable</PolicyVariableName>
                        </a:PolicyVariableKeys>
                        <ValueTypes>PolicyStringValue</ValueTypes>
                    </PolicyVariable>
                    <PolicyValue xsi:type="a:PolicyStringValueType">
                        <a:PolicyValueKeys>
                            <CreationClassName Key="true">PolicyStringValue</CreationClassName>
                            <PolicyValueName Key="true">Direction</PolicyValueName>
                        </a:PolicyValueKeys>
                        <StringList>in</StringList>
                    </PolicyValue>
                </PolicyCondition>
            </ConditionReference>
            <ConditionReference GroupNumber="1" ConditionNegated="false">
                <PolicyCondition xsi:type="a:fainSimplePolicyConditionType">
                    <a:PolicyConditionKeys>
                        <CreationClassName Key="true">fainSimplePolicyCondition</CreationClassName>
                        <PolicyConditionName Key="true">IPSourceCondition</PolicyConditionName>
                    </a:PolicyConditionKeys>
                    <EvaluationMethod>Match</EvaluationMethod>
                    <PolicyVariable>
                        <a:PolicyVariableKeys>
                            <CreationClassName Key="true">PolicySourceIPVariable</CreationClassName>
                            <PolicyVariableName Key="true">SrcIPVariable</PolicyVariableName>
                        </a:PolicyVariableKeys>
                        <ValueTypes>PolicyIPv4AddrValue</ValueTypes>
                    </PolicyVariable>
                    <PolicyValue xsi:type="a:PolicyIPv4ValueType">
                        <a:PolicyValueKeys>
                            <CreationClassName Key="true">PolicyIPv4AddrValue</CreationClassName>
                            <PolicyValueName Key="true">SrcIP</PolicyValueName>
                        </a:PolicyValueKeys>
                        <IPv4AddrList>147.83.106.0/24</IPv4AddrList>
                    </PolicyValue>
                </PolicyCondition>
```

```
            </ConditionReference>
            <ConditionReference GroupNumber="1" ConditionNegated="false">
                <PolicyCondition xsi:type="a:fainSimplePolicyConditionType">
                    <a:PolicyConditionKeys>
                        <CreationClassName Key="true">fainSimplePolicyCondition</CreationClassName>
                        <PolicyConditionName Key="true">IPDestCondition</PolicyConditionName>
                    </a:PolicyConditionKeys>
                    <EvaluationMethod>Match</EvaluationMethod>
                    <PolicyVariable>
                        <a:PolicyVariableKeys>
                            <CreationClassName Key="true">PolicyDestinationIPVariable</CreationClassName>
                            <PolicyVariableName Key="true">DestIPVariable</PolicyVariableName>
                        </a:PolicyVariableKeys>
                        <ValueTypes>PolicyIPv4AddrValue</ValueTypes>
                    </PolicyVariable>
                    <PolicyValue xsi:type="a:PolicyIPv4ValueType">
                        <a:PolicyValueKeys>
                            <CreationClassName Key="true">PolicyIPv4AddrValue</CreationClassName>
                            <PolicyValueName Key="true">DestIP</PolicyValueName>
                        </a:PolicyValueKeys>
                        <IPv4AddrList>128.40.40.0/24</IPv4AddrList>
                    </PolicyValue>
                </PolicyCondition>
            </ConditionReference>
        </PolicyCondition>
    </ConditionReference>
    <ActionReference ActionOrder="1">
        <PolicyAction xsi:type="a:EncActionType">
            <a:PolicyActionKeys>
                <CreationClassName Key="true">fainipvpnEncryptionAction</CreationClassName>
                <PolicyActionName Key="true">EncAction</PolicyActionName>
            </a:PolicyActionKeys>
            <EnforcementStrategy>Set</EnforcementStrategy>
            <IkeAuthentication>Authname</IkeAuthentication>
            <IkeEncryption>Encname</IkeEncryption>
            <IkeDHGroup>DHname</IkeDHGroup>
            <IkeTimeout>0</IkeTimeout>
            <IkeTrafficBasedExpiry>0</IkeTrafficBasedExpiry>
            <IPSecAuthentication>Authname</IPSecAuthentication>
            <IPSecEncryption>Encname</IPSecEncryption>
            <IPSecDHGroup>Dhname</IPSecDHGroup>
            <IPSecTimeout>0</IPSecTimeout>
            <IPSecTrafficBasedExpiry>0</IPSecTrafficBasedExpiry>
            <IkePeerAuthenticationMethod>1</IkePeerAuthenticationMethod>
        </PolicyAction>
    </ActionReference>
</a:fainPolicyRule>
```

8.3 XML EVENT MAPPING - CODE EXAMPLE

In this sub-section we describe, with an example, how the mapping of event classes is made to an XML-Schema. Moreover, the example of a concrete event in XML is given as well. This XML event is valid according to the XML-Schema given before. To better understand the syntax of XML and XML-Schema refer to [36] and [33] respectively.

8.3.1 XML-Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema targetNamespace="http://www.ist-fain.org/Schema" xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns="http://www.ist-fain.org/Schema">

    <xsd:notation name="FAIN-IPSec" public="http://www.ist-fain.org/schemas/FAIN/Event"/>

    <!-- contents of XML Schema document goes here -->

    <xsd:annotation>

        <xsd:documentation xml:lang="en">
```

```
    Event in XML Schema example.
    </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="fainEvent" type="fainEventType"/>
    <xsd:element name="EventKeys">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="CreationClassName">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="EventName">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="Key" type="xsd:boolean" use="fixed" value="true"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="fainEventType" abstract="true">
        <xsd:sequence>
            <xsd:element ref="EventKeys"/>
            <xsd:element name="ResourceIdentifier" type="ResourceIdType"/>
            <xsd:element name="CorrelatedEvents" type="CorrelatedEventsType" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="fainFaultEventType" abstract="true">
        <xsd:complexContent>
            <xsd:extension base="fainEventType">
                <xsd:sequence>
                    <xsd:element name="probableCause" type="xsd:string"/>
                    <xsd:element name="perceivedSeverity" type="xsd:string"/>
                    <xsd:element name="affectedResources" type="ResourceIdType" minOccurs="0"/>
                </xsd:sequence>
```

```
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="fainServiceDownEventType">
        <xsd:complexContent>
            <xsd:extension base="fainFaultEventType">
                <xsd:sequence>
                    <xsd:element name="relatedServiceComponents">
                        <xsd:simpleType>
                            <xsd:list itemType="xsd:string"/>
                        </xsd:simpleType>
                    </xsd:element>
                    <xsd:element name="AssociatedVE" type="ResourceIdType"/>
                    <xsd:element name="AdditionalInfo" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="fainRCReportEventType" abstract="true">
        <xsd:complexContent>
            <xsd:extension base="fainEventType">
                <xsd:sequence>
                    <xsd:element name="ResourceConsumption" type="ResourceConsumptionType"
maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="fainMonitorEventType" abstract="true">
        <xsd:complexContent>
            <xsd:extension base="fainEventType">
                <xsd:sequence>
                    <xsd:element name="MonitoredValue" type="MonitoredValuesType" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ResourceIdType">
        <xsd:sequence>
            <xsd:element name="RelativeId" type="RelativeIdType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="RelativeIdType">
        <xsd:sequence>
```

```
            <xsd:element name="KeyName" type="xsd:string"/>
            <xsd:element name="KeyValue" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CorrelatedEventsType">
        <xsd:sequence>
            <xsd:element ref="EventKeys"/>
            <xsd:element name="ResourceIdentifier" type="ResourceIdType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ResourceConsumptionType">
        <xsd:sequence>
            <xsd:element name="Resource" type="ResourceIdType"/>
            <xsd:element name="ConsumptionList" type="Consumption" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Consumption">
        <xsd:sequence>
            <xsd:element name="ConsumedValue" type="xsd:integer"/>
            <xsd:element name="unit" type="xsd:string"/>
            <xsd:element name="user" type="ResourceIdType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="MonitoredValuesType">
        <xsd:sequence>
            <xsd:element name="PropertyName" type="xsd:string"/>
            <xsd:element name="Value"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

8.3.2 XML Event Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT beta 4 build Jan 12 2001 (http://www.xmlspy.com) by Alexander Falk (Altova, Inc.) -->
<a:fainEvent xmlns:a="http://www.ist-fain.org/Schema" xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.ist-fain.org/Schema
Event_in_XML.xsd" xsi:type="a:fainServiceDownEventType">
    <a:EventKeys>
        <CreationClassName Key="true">fainServiceDownEvent</CreationClassName>
        <EventName Key="true">MulticastDown</EventName>
    </a:EventKeys>
    <ResourceIdentifier>
        <RelativeId>
            <KeyName>CreationClassName</KeyName>
```

```
            <KeyValue>MulticastService</KeyValue>
        </RelativeId>
        <RelativeId>
            <KeyName>ServiceName</KeyName>
            <KeyValue>MulticastGame</KeyValue>
        </RelativeId>
        <RelativeId>
            <KeyName>VECreationClassName</KeyName>
            <KeyValue>VE</KeyValue>
        </RelativeId>
        <RelativeId>
            <KeyName>VEName</KeyName>
            <KeyValue>ANN</KeyValue>
        </RelativeId>
        <RelativeId>
            <KeyName>ANNCreationClassName</KeyName>
            <KeyValue>ANNClass1</KeyValue>
        </RelativeId>
        <RelativeId>
            <KeyName>ANNCreationClassName</KeyName>
            <KeyValue>Node1</KeyValue>
        </RelativeId>
        <RelativeId>
            <KeyName>PDPCreationClassName</KeyName>
            <KeyValue>ServicePDP</KeyValue>
        </RelativeId>
        <RelativeId>
            <KeyName>PDPName</KeyName>
            <KeyValue>BarcelonaNode1</KeyValue>
        </RelativeId>
    </ResourceIdentifier>
    <probableCause>SoftwareProblem</probableCause>
    <perceivedSeverity>Critical</perceivedSeverity>
    <relatedServiceComponents>MulticastTopolocyManagerCode</relatedServiceComponents>
    <AssociatedVE>
        <RelativeId>
            <KeyName>VECreationClassName</KeyName>
            <KeyValue>VE</KeyValue>
        </RelativeId>
        <RelativeId>
            <KeyName>VEName</KeyName>
            <KeyValue>ANN</KeyValue>
        </RelativeId>
    </AssociatedVE>
```

```
</a:fainEvent>
```

9 APPENDIX D: DESCRIPTION OF TOOLS

9.1 INTRODUCTION

The most important characteristic of FAIN network management is that it is policy-based, which means that the tools selection should focus on policies. In order to make the description more clear, we use the basic policy based architecture given by IETF (shown in Figure 67), so as to ignore the detailed components given in the FAIN network management architecture, which is actually the expansion of IETF structure.

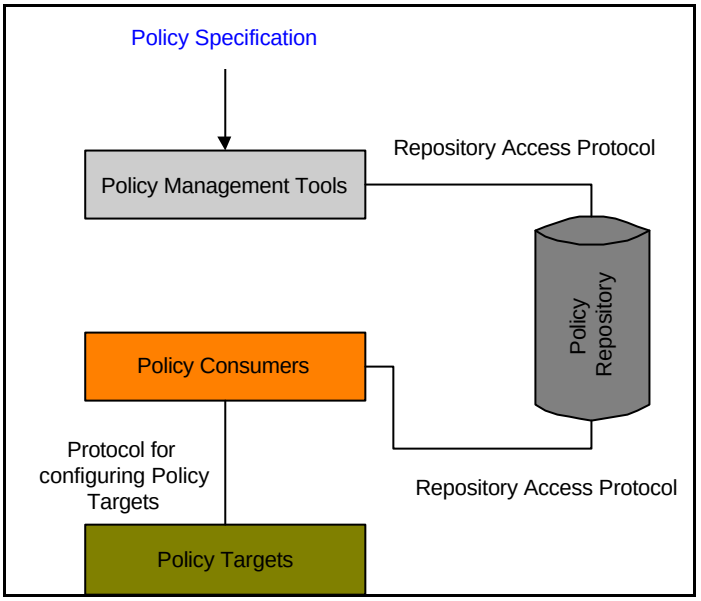

Figure 67- Basic PBNM Architecture given by IETF

The policy architecture as shown in Figure 67 suggests that these aspects as follow should be covered, from the software tool's point of view:

- How to represent policy
- How to store policy
- How to retrieve policy
- How to communicate between policy consumer (mainly PDP, therefore PDP will be used beneath) and policy target (supported by PEP).

Policy storage and retrieval are based on the same implementation technology, so they will be discussed together. Making a further step to Figure 67, a little more detailed can be given.

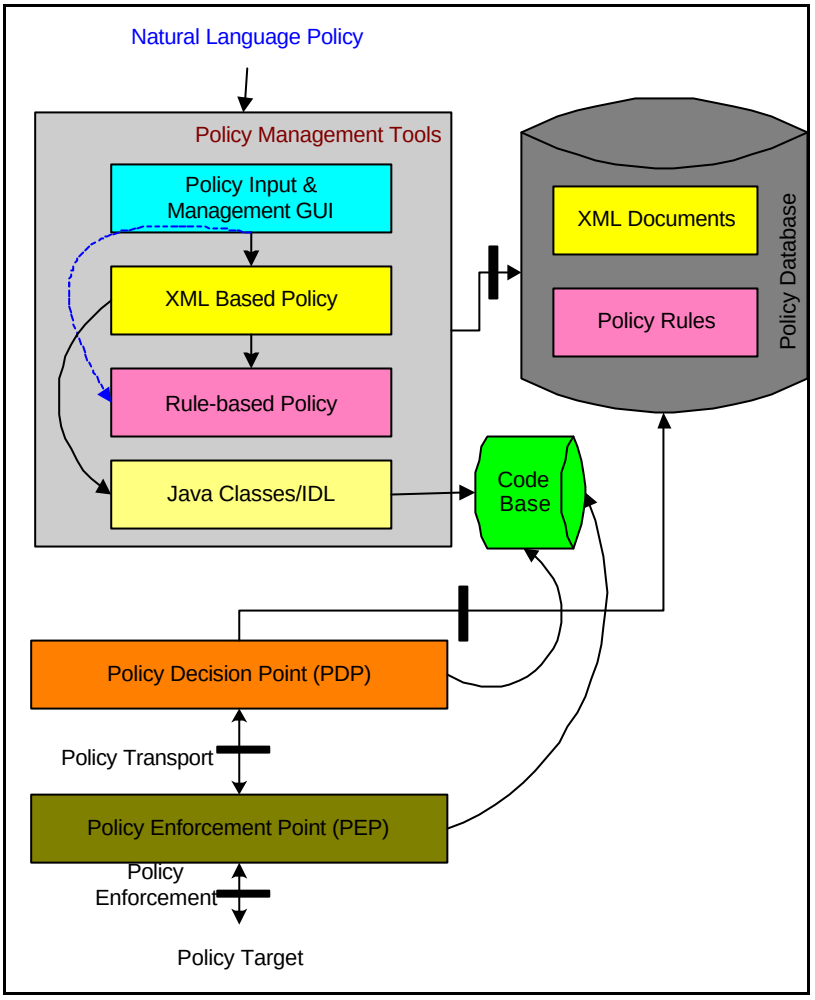

Figure 68 - Extension of PBNM architecture given in Figure 67

Therefore, four sections will be analysed in the coming paragraphs, which cover:

- Tools for policy input
- Tools for policy representation
- Tools for policy storage and retrieval
- Tools for policy transport
- Tools for policy enforcement

9.2 TOOLS FOR POLICY INPUT

In all the commercial PBNM products, policy input or provision is done through a graphical user interface (GUI). First the administrator selects the device or group of devices to which the policy will be applied. Then he can select from a menu of supported condition types and supply the necessary parameters. Then he can select from the supported actions. The PBNM tool identifies the selected devices, to check their capabilities and present to the user menus with only those conditions and actions that are supported by all the devices. The policy defined by the user has an “if (condition) then (action)” syntax. This kind of format is also used by the IETF.

The administrator can also create “roles” for network devices or interfaces and associate a set of policies with each role, so that when a new device is introduced in the network it can be assigned an existing role. In this way the configuration of the network is made easier.

As far as the author has known, there isn’t a well-known GUI designing tools specific for user-level network management policy input. But there are vast visualization software available for easily designing beautiful GUI.

This chapter only discusses the customer level policy input. XML-based policy input would be discussed in the next chapter.

9.3 TOOLS FOR POLICY REPRESENTATION

9.3.1 Policy specification

Various specification languages can be used in the context of policies, therefore various policy formats exist. In the terms of ease of use and final execution of policy, there can be (as shown in Figure 68):

- Natural Language format: supported by user-friendly GUI for customer to input or review policy in easy way. This depends on the design of GUI. This format must be mapping to the markup language format.
- Markup language format: this format can be processed and interpreted by a computer. But it has not mapped to the program code that can be executed directly. The most famous example of this kind of language is XML.
- Rule-based format: it interprets the policy as a sequence of rules, in which each rule is in the form of a simple condition-action pair. The rules are evaluated on specific triggers, such as the passage of time or the arrival of a new packet within the network. Policies specified in this fashion are easier to analyse than policies specified by a markup language or Java classes. IETF has chosen a rule-based policy representation in its specification. Therefore, rule-based policy will be focused in the rest of the document. But this format has to be mapped to Java classes or other program implementation, and also needs to be stored in an LDAP directory or database.
- Java classes: any policy object must be mapped to the Java object. This mapping should be done automatically.

Natural language format has been discussed above in Section 9.2 Java classes can be generated based on the interface IDL, which will be covered by other documents. Therefore, the following content of this section only focuses on the Markup language format and Rule-based format.

9.3.2 Tools for XML based policy representation

9.3.2.1 Brief Rationale for selecting XML as policy syntax

Various specifications for data formats exists, both in their syntax and semantic. To represent policies accurately and effectively, the syntax, semantic (metadata) and API for implementation have to be considered. This sub-section focuses on the syntax representation of policy, to which, widely used mark-up language will be used, as mentioned above.

Among all mark-up languages, eXtensible Mark-up Language (XML) is becoming increasingly adopted as a common syntax for expressing structure in data. XML is particularly suitable as a data representation mechanism for use in heterogeneous environments. Because XML is based upon an open industry standard, implementations of XML parsers exist for many platforms, and in many programming languages. Implementations are available on Unix and in C++ and Java. In addition, some parser implementations allow access to XML element tree from various different scripting languages and environments. XML's tree structures allow for context sensitivity and a more direct mapping to the object model than if flat parameter lists or logic expressions are used.

XML's significant advantages make it a good choice for policy representation for network management in FAIN. In policy-based network management environments, the XML representation of policies could be used to transfer policies between co-operating management platforms. These management platforms need not necessarily be running the same operating environments, but interoperability is enabled because of a common understanding of the XML representation of policy.

Furthermore, a wide variety of XML supporting tools exist to allow a fast PBNM development.

But XML does not provide any semantic for its document, and this is completely up to the users, which means FAIN needs to define the semantic of XML file used for carrying policy.

9.3.2.2 Semantics for XML-based policy

A document written in XML has to conform to some rules to be understandable and executable. These rules can basically be divided into two categories: Document Type Definition (DTD) and Schema. Schema is already a standard set by World Wide Web Consortium (W3C)[45], and is more powerful and flexible. As such, Schema mode will be used in FAIN. Within the Schema mode, there are still various specifications, of which Resource Description Framework (RDF)[46] and W3C Schema[47] are most famous.

9.3.2.2.1 RDF

RDF provides a common basis for expressing semantics in XML documents. It is a functional layer above XML. Applications, which allow programs to combine data logically, will be built using RDF (and therefore XML) and this will enhance the modularity and extensibility of the policy. One of the requirements of PBNM in FAIN is to allow huge amounts of policies to be stored in databases and existing applications to be put on the network, not just for user browsing, but also for machine understanding, i.e., searching, reasoning and analysing. This will need the help of metadata. RDF allows metadata applications to be combined, and to operate in a common way as the semantics that they share.

The RDF specifications also provide a lightweight ontology system to support the exchange of knowledge on the network, which is also a requirement for active networks. We intend to look for RDF specifications specific to PBNM, otherwise, a novel one would be developed.

Almost all the tools supporting XML also support RDF, so all these tools will be discussed together in sub-section **Error! Reference source not found.**.

9.3.2.2.2 W3C Schema

XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. Numerous information about W3C Schema is available on WWW, such as [47]. So we will not discuss it here in this document.

9.3.2.2.3 XML API

After the syntax and semantic of the policies have been determined, the next crucial issue to address is the implementation of these policies. Applications need to be developed in order to take an XML document and make its structure and content available to component that needs it, e.g., a PEP or even a node operating system (NodeOS) in the FAIN architecture. XML APIs are required for this task.

In general, there are two major types of XML APIs, i.e.,

- tree-based APIs
- event-based APIs.

A tree-based API compiles an XML document into an internal tree structure, and then allows an application to navigate that tree. A common example is the Document Object Model (DOM)[48].

On the other hand, an event-based API reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events - much like handling events in a graphical user interface. The typical one of this type is Simple API for XML (SAX)[49].

While DOM APIs are useful for a wide range of applications, they often put a great strain on system resources, especially if the document is large. Under very controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem. Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

SAX API provides a simpler, lower-level access to an XML document. Documents much larger than available system memory can be parsed, and one can construct specific data structures using callback event handlers.

The selection of SAX or DOM should depend on the applications or scenarios, therefore both of them may be used in FAIN.

9.3.2.3 Tools for XML Processing

9.3.2.3.1 XML Browser

When working with XML, we will need to view XML documents. The tool for this purpose is usually called XML browser. XML browsers are generally driven by style sheets, and are tree-based. They can be generic ones such as IE5 and Netscape, or application specific ones. XML editors, as shown below, usually also have this functionality.

9.3.2.3.2 XML Editor and XML Spy

For input of XML document, the ordinary text editor works. But if special XML editors are used, less mistakes will be made. Many XML editors are sensitive to Schemas and/or DTDs and so can enable user to easily produce well-formed and valid XML documents. Cooktop is a good option, but XML Spy[50] is more powerful. More XML tools can be found on XMLSOFTWARE[51], which aims to provide well-organised information, resources, especially software tools on XML.

XML Spy is the first true Integrated Development Environment for XML that includes all major aspects of XML in one powerful and easy-to-use product, which can be downloaded free of charge. XML Spy is centered around a professional validating XML editor that provides five advanced views on your documents:

- An Enhanced Grid View for structured editing.
- A Database/Table view that shows repeated elements in a tabular fashion.
- A Text View with syntax coloring for low-level work.
- A graphical XML Schema designs view.
- And an integrated Browser View that supports both CSS and XSL style-sheets.

9.3.2.3.3 XML Parser and Sun JAXP

A parser or processor takes an XML document and makes its structure and content available to an application. Often via a standard interface like SAX or DOM. Basically, XML Parser can be divided into two categories: application/product specific XML parser, such as Oracle XML parser for C/Java, and generic XML parser.

XML parser also language-oriented. So there are XML parser for C, XML parser for Java, XML parser for COBOL, XML parser for PL/SQL, etc.

Generic XML parsers are used in FAIN[52], of which Sun Java API for XML Processing (JAXP) [98] is highly appreciated. JAXP enables applications to parse and transform XML documents using a pure Java API that is independent of a particular XML processor implementation. Depending on the needs of the application, developers have the flexibility to swap between XML processors (such as high performance vs. memory conservative parsers) without making application code changes. Thus, application and tools developers can rapidly and easily XML-enable their Java applications.

The reference implementation uses Crimson [54], which was derived from the Java Project X parser from Sun, as its default XML parser and Xalan [110] as its default XSLT engine. Therefore, JAXP 1.1 also supports DOM Level 2 and SAX version 2.0. However, the pluggable architecture of JAXP allows any XML conformant implementations to be used.

9.3.3 Tools for Rule-based policy representation and reasoning

9.3.3.1 Support to XML rule representation in ILOG Rules

A rule-based commercial software toolkit called ILOG Rules will be evaluated in FAIN. ILOG Rules[41] provides a common environment for building and managing enterprise-wide business-rule applications. ILOG Rules provides a common set of tools which includes:

- Rule Builder: integrated development environment for developing and debugging business-rule applications.
- Rule Language: customisable and extensible business-rule language, placing business-rule power in the hands of business users.
- Rule Editor: powerful, adaptable Web-enabled and JavaBean components that can be embedded in applications.

The most important thing is that ILOG Rules also provides an XML rule representation, which allows sharing of rules between applications, and generating and processing executable rules using standard XML tools, such as XSTL, SAX2, DOM, and JDOM. The rule engine API supports parsing the XML rule representation. This functionality will be highly used in FAIN.

9.3.3.2 Support of reasoning in ILOG Rules

ILOG rule engines incorporate temporal reasoning and transition monitoring into rules. Temporal reasoning may be used to designate a waiting period of a specified duration or until a specified time within rules. Transition monitoring can be used to test whether certain conditions remain true for either a specified duration or until a specified time.

9.4 TOOLS FOR POLICY STORAGE AND RETRIEVAL

9.4.1 Overview

After the definition of a new policy (can be in other forms besides XML), the policy is submitted and stored in the policy repository or database (DB).

For the implementation of the policy repository, there can be various solutions, which can be summarised as three categories: directory accessed by LDAP protocol, database accessed by JDBC/ODBC such as Oracle and Sybase, or a plain text file. In every above case, different approaches to access them can be applied. Here, some methods or tools supposed to be used in FAIN are discussed.

9.4.2 LDAP-based directory and JNDI

In FAIN, X.500 based directory and LDAP would be used. Each policy will be an entry in the directory. However, many different representations of the Policy can be stored for a single Policy entry. Storing the policies uses an LDAP Schema i.e. storing the individual attributes of the policy using a suitable schema so that search for policies can be based on their attributes. The name of the policy is most important, which might be stored separately.

LDAP-based directory software tools exist on many platforms, both for languages and operating systems. As far as Java language is used, Sun JNDI [101] is a very good selection. Java Naming and Directory Interface (JNDI) can also provide applications written in the Java programming language with a unified interface to multiple naming and directory services, and more powerfully. JNDI enables seamless connectivity to heterogeneous enterprise naming and directory services. Developers can build powerful and portable directory-enabled applications using this industry standard. Java 2 SDK 1.3 includes JNDI, which supports LDAP v3.

9.4.3 Relational Database Management System and JDBC

Policy can also be stored in traditional RDBMS such as Oracle, Sybase, etc. As far as Java programming language is used, JDBC can be used for the interface. If other languages or platforms are preferred, ODBC can serve as the interface. Both of them focus on executing raw SQL statements and retrieving their results. Since Java is widely used and there is the bridge from JDBC to ODBC, we just discuss JDBC here.

The JDBC API provides universal data access from the Java programming language. Using the JDBC 2.0 API, programmer can access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base on which tools and alternate interfaces can be built.

The JDBC 2.0 API includes two packages: the java.sql package, known as the JDBC 2.0 core API, and the javax.sql package, known as the JDBC Standard Extension. The Java 2 SDK, Standard Edition, includes the JDBC 2.0 core API and the JDBC-ODBC Bridge. The Java 2 SDK, Enterprise Edition, includes the JDBC 2.0 core API and also the JDBC 2.0 Standard Extension.

To use the JDBC API with a particular database management system, a JDBC technology-based driver is needed to mediate between JDBC technology and the database. JDBC drivers can be of different types. As far as we know at this time, there can be four categories: JDBC-ODBC bridge plus ODBC driver, Native-API partly-Java driver, JDBC-Net pure Java driver, and Native-protocol pure Java driver. The last two driver categories are preferred to access databases from JDBC because of their high performance.

9.5 TOOLS FOR POLICY TRANSPORT

9.5.1 Common Open Policy Service (COPS) and Vovida.org

The IETF Resource Allocation Protocol (RAP) WG has developed the Common Open Policy Service (COPS) [102] as a policy protocol for use in PBN management systems. COPS is a query and response protocol that can be used to exchange policy information between a policy server and its clients.

Vovida.org is a communications community site dedicated to providing a forum for open source software used in network environments. It provides quite a lot of application software on Session Initiation Protocol (SIP), Cisco SIP proxy servers (CSPS), RTP (Realtime Transport Protocol), Media Gateway Control Protocol (MGCP), etc. It also provides support for Common Open Policy Service (COPS). Here in this document focus will be given to COPS, which is what we will use in PBNM.

All COPS messages are supported. It also includes COPS extentions to support Policy Provisioning (COPS-PR).

Its shortcoming is that it doesn't support Ipv6 addresses yet.

9.5.2 Simple Object Access Protocol (SOAP) and Apache SOAP

SOAP[58] is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of protocols. All SOAP messages are encoded using XML.

Apache SOAP [59], is an implementation of the W3C SOAP, is recommended in FAIN. It is based on, and supersedes, the IBM SOAP4J implementation[60]. Its main features are described as follow [61]:

- Supports most of the SOAP v1.1 specification Provides server-side infrastructure for deploying, managing and running SOAP enabled services
- Provides client-side API for invoking SOAP services
- Release includes full source under the Apache Software License
- Supports three encoding styles: SOAP v1.1 Encoding, Literal XML and XMI.
- XMI encoding (available when using Java 1.2.2) supports automatic marshalling and unmarshalling of arbitrary objects
- SOAP encoding: built-in support is provided for encoding/decoding primitive types, Strings, arbitrary JavaBeans (using reflection) and 1-dimensional arrays of these types. For other types user can hand-write encoder/decoder and register with XML-SOAP runtime.
- Literal XML encoding: allows one to send XML elements (DOM org.w3c.dom.Element objects) as parameters by embedding the literal XML serialization of the DOM tree. No code needs to be written to support this (see the addressbook demo to see a sample use of it).
- Supports messaging and RPC over two transports: HTTP and SMTP Supports authoring services in scripting languages

9.5.3 XML-RPC

A relatively simpler protocol - XML-RPC [40] can also be a option for policy transport between PDP and PEP.

XML-RPC is a Remote Procedure Calling protocol that works over the Internet, using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned. An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML. At present, XML-RPC is used for the policy transport between network level and element level.

9.5.4 Mobile agent technology and Grasshopper

The mobile agent paradigm intends to bring an increased performance and flexibility to distributed systems by promoting "autonomous code migration" (mobile code moving between places) instead of traditional RPC (remote procedure call). With code migration, the actual code or script moves from place to place and executes locally, achieving lower latency, little need for remote interactions and highly flexible control. Mobile agent technology is widely used in telecom and network management. They are very effective as they can take over the burden of the complex interaction mechanisms between different network players, such as negotiations or new service injection. Mobile agents can easily represent one of the business roles such as backbone operator, access provider, service provider or end-user, and act on their behalf, based on established policies.

Based on these advanced features, mobile agent can be used for the policy transport between PDP and PEP.

Examples of mobile agent platforms are Odyssey (General Magic), D'agent/Agent TCL (Dartmouth College), Voyager (ObjectSpace), Aglets (IBM) and Grasshopper (IKV++), of which Grasshopper will be used in FAIN.

Grasshopper [42] is a mobile agent development and runtime platform which is built on top of a distributed processing environment. This achieves an integration of the traditional client/server paradigm and mobile agent technology. Grasshopper is implemented in Java, based on the Java 2 specification. Most importantly, Grasshopper has been designed in conformance with the first mobile agent industry standard given by OMG, namely the Object Management Group's Mobile Agent System Interoperability Facility (MASIF) [62], which allows interoperability of different mobile agent platforms and the deployment of mobile agents on CORBA environments. In addition, the latest Grasshopper version is also compliant with the specifications of the Foundation for Intelligent Physical Agents (FIPA) [44]. Grasshopper is also the agent platform of choice in multiple international research projects within the European CLIMATE (Cluster for Intelligent Mobile Agents for Telecommunication Environments).

9.6 TOOLS FOR POLICY ENFORCEMENT

9.6.1 COPS and SOAP

COPS also provides the functionality of policy enforcement in some extents, which has been shown in Section 9.5 So does SOAP. Here emphasis is given to the mobile agent based SNMP policy enforcement. Common Management Information Protocol (CMIP) can also be used by mobile agents.

9.6.2 AdventNet SNMP

AdventNet SNMP V3.2 is a set of Java tools for creating cross platform Java and Web-based SNMP network management applets and applications. The package can be used to develop SNMP management applications to manage SNMPv1, SNMPv2c and SNMPv3 agents (i.e. the management applications can be multi-lingual) and talk to agent systems using any of the three versions of the SNMP protocols at the same time.

AdventNet SNMP V3.2 includes Java classes that implement:

- SNMP communication for SNMPv1, SNMPv2c and SNMPv3 protocols
- SNMPv3 security as defined in User based Security Model (USM) and View based Access Control Model (VACM) definitions
- MIB support for both SMIv1 and SMIv2 formats so that Java management applications can take advantage of the information contained in the MIB files
- SNMP Applet Server (SAS) to facilitate communication between applets and managed devices where direct communication is prohibited due to applet security policies

- SNMP Beans Components like SnmpTarget, SnmpPoller, SnmpTrapReceiver etc. that provide enhanced functionality and for use in Java IDE tools.
- RMI and CORBA access to SNMP API for distributed computing support.

9.6.3 Grasshopper based AdventNet SNMP

Based on the given policies, Grasshopper mobile agents bearing the PEPs that are in charge of corresponding policy as destinations are created automatically. These mobile agents migrate themselves to the specific PEPs to enforce the policy.

After arriving at the PEP, mobile agent, also served as SNMP wrapper at this moment, uses SNMP to fulfil the policy.

After getting the return code of SNMP command execution, mobile agent goes back to PDP to inform the result, then removes itself automatically.

Here all mobile agent execution environment (i.e. Grasshopper Agency) needed for mobile agent life cycle is pre-installed on every PEP. If there is not mobile agent EE on the destination EE, active node mechanism such as ABLE++ can be used to transfer and setting up the MA EE. Agency resides in a machine next to that element.

A Region Server, which provides the agents with the necessary information and location of the respective agencies, is also needed.

In addition, wrappers for the SNMP protocol had to be developed to enable the agents to communicate to the managed elements via SNMP. For simplicity, the functionality of wrapper is also implemented by mobile agent, in which Advent SNMP Driver is used.

9.7 CONCLUSION ON TOOLS

Based on the fundamental functionality of policy based network management, this section has proposed an overall description of tools used or supposed to be used in FAIN, which mainly includes: tools for policy input, tools for policy representation, tools for policy storage and retrieval, tools for policy transport, and tools for policy enforcement. The relevant software products are also proposed and discussed from a more abstract view.

