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

D14 – Overview FAIN Programmable Network and Management Architecture - Draft

Editor:	**Spyros Denazis/ Alex Galis**
Document No:	**WP3-HEL-056-D14-FAIN_Overview.doc**
Contribution File Name:	**WP3-HEL-056-D14-FAIN_Overview.doc**
Version:	**2.0**
Company:	**HEL/UCL**
Date:	**Monday 12th May 2003**
Distribution:	**WP3, WP4, WP5**
Dissemination:	**CO**

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
HEL	Partner	United Kingdom
HIT	Partner	Japan
SAG	Partner	Germany
ETH	Partner	Switzerland
FHG/FOKUS	Partner	Germany
IKV	Associate Partner to FHG/FOKUS	Germany
INT	Associate Partner to FHG/FOKUS	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)
Hitachi Europe Ltd.	(HEL)
Hitachi Ltd.	(HIT)
Siemens AG	(SAG)
Eidgenössische Technische Hochschule Zürich	(ETH)
Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	(FHG/FOKUS)
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 5738
Fax +44 (0) 207 388 9325
E-mail: a.galis@ee.ucl.ac.uk

Authors

Elisa Boschi (FhG) - Contributor
Celestin Brou (FhG) - Contributor
Stamatis Karnouskos (FhG) - Contributor
Cornel Klein (SAG) - Contributor
Spyros Denazis (HEL) - Contributor/ Editor
Alex Galis (UCL) - Contributor/ Co-editor
Lukas Ruf (ETH) - Contributor
Epi Salamanca (UPC) - Contributor
Joan Serrat (UPC) - Contributor
Marcin Solarski (FhG) - Contributor
Julio Vivero (UPC) - Contributor

Change History

Ver.	Date	Authors	Comments
0.1	11.03.2003	Spyros Denazis (HEL)	Initial ToC
0.2	11.04.2003	Spyros Denazis (HEL) Cornel Klein (SAG) Epi Salamanca (UPC) Joan Serrat (UPC) Marcin Solarski (FhG) Elisa Boschi (FhG) Julio Vivero (UPC)	• Initial Contribution of FAIN Reference Architecture & FAIN Nodes sections (HEL) • Initial Contribution of the Business Model (SAG) • Initial Contribution of PBNM (UPC) • Initial Contribution of ASP (FhG) • Initial Contribution of the FAIN testbed (FhG) • SoA Analysis (UPC)
0.3	10.05.2003	Alex Galis (UCL)	• Networking Issues • Scenarios summary • Conclusions • Future developments • General editing
1.0	12.05.03	Alex Galis (UCL)	• General edititing •Introduction
2.0	12.05.03	Alex Galis (UCL)	•Final editing

Table of Contents

Table of Figures

1 EXECUTIVE SUMMARY AND INTRODUCTION

Deliverable D14, which is additional to the project Technical Annex, is planned for the end of June 2003. It is released as a draft in May 2003 for the project review.

D14 represents an overview of the architectural results of the FAIN project, which were developed and published in three annual series of project deliverables as follows:

Topic	Project Year 1 Deliverables	Project Year 2 Deliverables	Project Year 3 Deliverables
Active Networks Requirements & Enterprise Model	D1		
FAIN Active Node Architecture	D2	D4	D7
FAIN Network Management and Active Service Provision Architecture	D3	D5	D8
FAIN Testbed, Scenarios and Evaluation		D6	D9, D40

These annual series of project deliverables were produced as phased revisions: Concept revisions and /or Technical revisions.

Concept revisions refer to the main architectural concepts outlined in year 1 or year 2 deliverables as they needed more focus in some cases or lacked completeness in the previous version of the deliverable. In the year 3 deliverables we have revisited the concepts and described them from a different viewpoint while making the necessary references to the corresponding implementation, thereby adding more depth in their description by connecting them with experimental proofs.

Technical revisions refer to the implementation of the FAIN Active Network and Management architecture, which resulted in modifications, or extensions of the initial version of the architecture description as well as the particular choice of technologies and the engineering aspects thereof.

More specifically, ***Section 2*** *(Review of Active and Programmable Networks Technologies*) provides an analysis of the state of the art in active and programmable networks, in particular the state of the art of Node Operating Systems (Node-OS) for active and programmable networks, and the Execution Environments research.

Section 3 *(Review of Management Systems Technologies)* provides an analysis of management of active and programmable networks research and in particular on network management (i.e. we focus in policy-based management of active networks) and active service provisioning.

Section 3 *(FAIN Enterprise Model)* provides a definition of the FAIN enterprise model consisting of actors (i.e., persons, companies, or organisations of any kind that own, use, support and/or administrate parts of an active networking system), various business roles (i.e. behaviour, properties, and responsibilities assigned to an actor) and the contractual relationships between these roles (i.e. the reference points, which determine the interactions between the actors in the respective roles).

Section 5 *(FAIN Reference Architectural Model)* provides a reference architecture, which underpins the FAIN work and describes how the ingredients identified in Section 4 can be combined synergistically to build next generation Network Elements capable of seamlessly incorporating new functionality or being dynamically configured to change their behaviour according to new service requirements.

Section 6 *(FAIN Networking Architecture)* summarises the networking concepts, which were the result of the project's intention to design a system that is flexible and interoperable. It describes the Network Elements and Network Management components, which are part of the FAIN networking system.

Section 7 *(FAIN Active Node)* summarises the main components of the FAIN Active Router: Virtual Execution Environment Manager (VEM), *Demultiplexing facilities* (DEMUX), *Resource Control Facilities* (RCF), *Security Facilities* (SF), Execution Environments (EE) types (Java EE, a control EE: SNMP EE and a high-performance EE based on Linux Kernel, called PromethOS).

Section 8 *(FAIN PBNM Management)* summarises the main architecture and components of FAIN Policy-based management system. The FAIN PBNM management architecture is designed as a hierarchically distributed architecture, consisting of two levels (two-tiered architecture): the network management level, which encompasses the Network Management System (NMS) and the element management level, which encompasses the Element Management System (EMS).

Section 9 *(FAIN Active Service Provisioning)* summarises the project's Active Service Provisioning facility for deploying active services in the FAIN network. Active service deployment is considered as a process of making a service available in the active network so that the service user can use it. The deployment process consists of a number of preparatory activities before the service operation phase, including: releasing the service code, distributing the service code to the target location, installing it and activating it.

Section 10 *(FAIN Testbed)* provides an overview of the operational FAIN active testbed (the structure of the testbed, the location of different facilities and components, description of the type of nodes running at various sites), which serves as a permanent experimental network for active network technologies up to the end of the FAIN project and possibly beyond.

Section 11 *(FAIN Scenarios)* provides an overview of the application scenarios designed and demonstrated in the FAIN project to run on the project testbed. They include: DiffServ Scenario, WebTV Scenario, Web Service Distribution Scenario, Video on Demand Scenario, Mobile FAIN Demonstrator Scenario, Managed Access Scenario and Security Scenario.

Section 12 *(Concluding Remarks*) concludes on the functionality deployed in the FAIN testbed and demonstrations and highlights the key achievements of the project.

Section 13 *(Future Developments in Programmable Service Networks)* identifies some of the key research and development issues, which are underpinning the migration from active networks towards programmable service networks.

Section 14 *(References for the SoA Review) and* ***Section 15*** *(Other References)* identifies some of the key references used in the FAIN project.

2 REVIEW OF ACTIVE AND PROGRAMMABLE NETWORKS TECHNOLOGIES

Since the mid-nineties, there has been a proliferation of new concepts, architectures and technologies that have contributed to a networking paradigm shift. The motivation behind this shift has been the rapid and autonomic service creation, deployment, activation and management combined with context customisation and customer personalisation. Such motivation can be traced in different organizations, fora, and research activities as well as market forces.

Among these new networking technologies and paradigms, the introduction of programmability in the network elements (switches, routers etc) has emerged as de facto technology for the rapid deployment and customisation of new services. Strongly pushed forward by DARPA and other organizations and standardisation bodies many projects have studied different active and programmable network architectures. In this chapter we analyse different projects that have researched active and programmable networks, and which encompass the range of possible solutions. At the end of the chapter we summarise the direction that current research and standardisation activities seem to be taking in the short to medium term.

2.1 STATE OF THE ART

In this section we analyse the state of the art in active and programmable networks. We have divided the section in two parts: the first one dealing with the state of the art of Operating Systems (OSs) for active and programmable networks, and the second one dealing with the state of the art of Execution Environments research.

2.1.1 OSs for Active and Programmable Networks

In the area of active networks, two distinct approaches have been around for quite some time: on the one hand, the pure capsule [2], [3] and [4], and on the other hand the router plug-in [5] approach. While the former can be seen as an extreme in terms of how program code is injected into the network, the latter resembles a transition from upgradeable router architectures towards programmable, high-performance active network nodes.

With the capsules approach every packet carries code that is executed at each node. For example, the functionality provided in the capsules may handle packet-routing requests or payload modifications to be carried out on a node. Plug-ins are code components that are installed out-of-band on an active node. Usually, they serve for long-lived flows, i.e. they extend the base functionality for a large set of packets. Thus, the overhead is smaller with the plug-in approach than with the capsule approach.

Capsules commonly make use of a virtual machine that interprets the capsule's code to safely execute it on a node. In order to ensure security, the virtual machines must restrict the address space a particular capsule might access, thus restricting the application of capsules. We expect network links to be 10Gbps or faster in the near future. With an optimistic average packet size [6] or 512 bytes for IP traffic, a router has to process 2.6 million packets per second on every port, or less than 380 nanoseconds per packet. Even if we assume that a significant fraction of the packets forwarded do not require active processing and can be handled in hardware, it seems obvious that active network architectures based on virtual machines are not well suited to a multi-gigabit scenario. However, they may be relevant for tasks such as network management where only sporadic configuration and management tasks are to be carried out.

Multimedia communication over heterogeneous network infrastructures will drive the requirements for particular network protocols with per-customer adaptation. A classical example is the situation in which a customer receives a multimedia stream that needs data adaptation such as down-scaling at the gateway between the Internet and the mobile device.

A very important observation is that the deployment of multimedia data sources and applications (e.g. real-time audio/video, IP telephony) will produce longer-lived packet streams (flows) with more packets per session than is common in today's Internet. Especially for these kinds of application, active networking with the plug-in-approach offers very promising possibilities: media gateways, data fusion and merging, and sophisticated application-specific congestion control.

We are convinced that network code will never be programmed by common users but rather by specialists [7]. Service code providers are expected to deliver components that are freely inter-connectable. These components are stored on a central repository and fetched by the management systems to be deployed and configured on an active network node. Even though programmed by specialists, components may be erroneous and thus compromise node stability and security.

Based on these observations and assumptions, the following conditions may be defined:

- Management and control functionality that is not time-critical can be carried out in virtual machines without sacrificing overall node performance.
- Deployment and configuration of decoupled service-components is allowed to require extended, time-consuming steps.
- Resource Control is required for safely multiplexing physical resources on a node.
- High-performance active network node architectures are required for flexibility of application-specific code processing at link-speed of routers.

To implement these requirements, a NodeOS for active networks providing the outlined functionality is required. To compare the approach we pursued in FAIN in this area, we focus on high-performance NodeOS that are based on the plug-in approach.

In this area, a basic distinction must be made between add-ons to widespread used legacy operating systems like NetBSD [8] and Linux [9] or even Microsoft Windows [10], and proprietary NodeOS that have been designed and implemented from scratch. We give a short overview of current research and provide a very brief discussion of the goals followed by the proposed candidates.

Add-ons to Linux, NetBSD and MS Windows:	*Proprietary NodeOS:*
1. Pronto	1. Scout
2. Bowman	2. Nemesis
3. SILK -- Scout in the Linux Kernel	3. Exokernel
4. PromethOS	4. Moab/Janos
5. Crossbow/ANN	
6. Lara++	

Table 1 - OSs summary table

2.1.1.1 Proprietary NodeOS

- **Scout**

Scout [86], as based on xkernel v2, implements the path-abstraction. It is a single address space research operating system without resource control mechanisms.

- **Nemesis**

Nemesis [88] is a research operating system for multi-media, low-latency communication. It introduced the fbuf-structure to allow inter-process communication with zero-copy mechanisms. Nemesis was extended by RCANE – A Resource Controlled Active Network Environment -- for resource control issues.

- **Exokernel**

Exokernel [89] multiplexes physical resources by providing a so-called library operating system that exports interfaces that are as close to the hardware as possible thus introducing as little as possible overhead. A clear disadvantage is code redundancy and the requirement to re-implement basic functionality found in legacy operating systems.

- **Moab/Janos**

Moab [90] is a research operating system based on the OSKit. It exports interface as required for the Janos [91] operating system. Janos creates a Java Virtual Machine with resource control in mind.

2.1.1.2 Add-ons to Linux, NetBSD and MS Windows

- **Linux: Pronto**

Pronto [84] provides a framework for node programmability. It is based on Linux and runs in kernel space. It follows the plug-in approach by providing an own execution environment.

- **Linux: Bowman**

Bowman [46] implements the Active Network Interface Specification in user space of Linux.

- **Linux: SILK -- Scout in the Linux Kernel**

SILK [85] extends the Linux kernel by providing the path-abstraction in the kernel space. By SILK, the Scout architecture has been ported to a widely used operating system. Basically the architecture can be compared to the Linux netfilter architecture for packet mangling.

The NodeOS interface specification is mapped from user space on the Linux kernel by SNOW. Scout and SILK intercept the system call interface of Linux for communication with the user space; the GENI interface on the other end is connected to the input port at the Linux network stack. However, SILK does not provide resource control mechanisms. Vera provides an interface to the Intel IXP1200 network processor. It is attached to SILK by providing a virtual device in Linux. By VERA programming of the microEngines in the IXP1200 is achieved.

- **Linux: PromethOS v1**

PromethOS [87] v1 extends the standard netfilter architecture of Linux by adding at run-time programmability and extensibility to the Linux kernel for unknown components. Following the interfaces of netfilter strictly, the whole framework is inherently portable. Performance of the PromethOS framework is comparable to the standard Linux networking environment; only one additional hash-table lookup has been introduced to schedule the PromethOS plug-ins.

- **NetBSD: Crossbow/ANN**

Crossbow [76] follows the ideas of Scout and the path-abstractions. Flows can be bound to plug-in chains. It forms the conceptual basis of PromethOS. No resource control mechanisms are foreseen. Due to its implementation, Crossbow is deeply bound to a specific release of NetBSD.

- **MS Windows: Lara++**

2.1.1.3 LARA++

[10] provides an active node operating system that is implemented on Windows. It provides an execution environment (called a processing environment) into which active components are loaded. These components are interconnected to form a graph of similar to the concepts of Scout.

Summary

Summarising the overview, we clearly identify a lack of resource controllable, widespread, flexible and high-performance node operating systems (NodeOS). Support of currently available and future network processors as well as mapping strategies for components are required, too. These issues are currently addressed in the context of PromethOS.

Based on this brief overview of operating systems suitable for active networking, the following issues can be extracted that could be addressed standardisation in future or other research projects:

- Service Composition and Control: The interface to trigger the service composition as well as its configuration should be standardised such that management and control components become re-usable for different NodeOS.
- Interfaces to specify resources in a very fine granular manner are required.
- Interfaces for inter-component and inter-EE communication are mandatory to ease the creation of decomposed services.

2.1.2 Execution Environments

The Active Network (AN) community has designed and embraced an architectural framework for Active Networks [11] which defines a three layer stack on each active node, of which the second one is represented by one or more Execution Environments (EE) that define a programming model for writing Active Applications (AA). Several well-known efforts that provide EEs based on this approach include ANTS, PLAN, SNAP, CANES and ASP. Finally a management EE (e.g. SENCOMM [38]) and a management AA are usually initiated at bootstrap in order to offer management services at EE or AA level respectively.

2.1.2.1 EE's review

- **ANTS**

ANTS [40] is a Java-based toolkit for constructing an active network and its applications. ANTS is based on an aggressive "capsule" approach in which code is associated with packets and run at selected IP routers that are extensible. The latest version of ANTS (version 2.x) relies on the Janos Java NodeOS.

- **PLAN**

PLAN [41] is a functional, scripting language (based on CaML) with limited capabilities, designed to execute on routers. The fundamental construct in the language is one of remote evaluation of delayed functional applications [42]. PLAN is designed to be a public, authentication-free layer in the Active Network hierarchy, therefore it has limited expressive power in order to guarantee that all programs will terminate. PLAN can also be used as a "glue" layer that allows access to higher-level services (which may or may not require authentication). This combination allows much of the flexibility of active networking, without sacrificing security.

- **SNAP**

SNAP [43] is an active networking system where traditional packet headers are replaced with programs written in a special-purpose programming language. The SNAP language has been designed to be practical and with focus on efficiency, flexibility and safety. Compared to PLAN, SNAP offers significant resource usage safety and achieves much higher performance at the cost of flexibility and usability. A PLAN to SNAP compiler has also been developed [44].

- **CANES**

CANES [45] seeks an approach to active networks that supports high performance while also permitting dynamic modification of network behaviour to support specific applications and/or provide new services. The CANES Execution Environment runs on the Bowman [46] implementation of the NodeOS specifications and is specifically built for composing services within the network [47].

- **ASP**

ASP [37] is implementing the "strong EE model" by offering a user level operating system to the AAs via a Java based programming environment. The underlying capabilities of NodeOS and Java are enhanced (e.g. usage of Netiod [48] to replace the socket interface) and complex control plain functionality such as signalling and network management can be realised. Customised security Manager, out-of-band code loading, routing of active packets via IP and virtual connectivity (UDP/IP tunnelling) are some more features of the ASP EE.

- **FAIN EE**

One of the key concepts defined by the FAIN architecture is the EE and VE (as a grouped EEs). In FAIN, drawing from an analogy based on the concepts of class and object in object-oriented systems, we distinguish EEs between the *EE type* and the *EE instances* thereof. An EE type is characterised by the programming methodology and the programming environment that is created as a result of the methodology used. The EE type is free of any implementation details. In contrast, an EE instance represents the realisation of the EE type in the form of a runtime environment by using specific implementation technology e.g. programming language and binding mechanisms to maintain operation of the runtime environment. Accordingly, any particular EE type may have multiple instances while each instance may be based on different implementations. Such a distinction allowed us to address the principles that must govern next generation NEs and the properties that they must possess, separately from the issue of how to build such systems.

The programming methodology that was used as part of the FAIN EE type was the building block approach according to which services break down into primitive, distinct blocks of functionality, which then may be bound together in meaningful constructs. To this end, services can be rebuilt from these primitive forms of functionality, i.e. the building blocks, while building blocks may be reused and combined together in a series of different arrangements as dictated by the service itself. In FAIN we have built two different EE instances, a Java EE and a Linux kernel-based EE, of this particular EE type [18].

The FAIN architecture also allows EEs to reside in any of the three operational planes namely, transport, control and management while they may interact and communicate with each other either across the planes or within a single plane. In fact, it is not the EEs that communicate but rather distributed service components hosted by them, a subset of the deployed network services which can be accessed by applications or higher level services by means of the network API they export. EEs (instances) are the place where services are deployed. Services may well be extensible in the sense that the programming methodology and the corresponding environment (EE type) support service extension while they can access services offered by other EEs to achieve their objectives and meet customer demands. For example a service uses the code distribution mechanism to download its code extensions. The extension API then becomes part of the overall service interface.

Furthermore, FAIN separates the concept of the EE from that of the Virtual Environment (VE). We argue that the concept of an EE as defined previously and that of a VE are orthogonal to each other. In fact a VE is an abstraction that is used only for resource management and control. Therein services may be found and may interact with each other. From the viewpoint of the operating system, the VE is the principal responsible for the consumption and use of resources, the recipient of sanctions in the event of policy violations and the entity that can legally receive authorisation when other services access the control interfaces. Similar conclusions may be found in [15], [34]. In other words, a VE provides a container in which services may be instantiated and used by a community of users or groups of applications, while remaining isolated from components residing in different VEs. Within a VE many types of EEs with their instances may be combined to implement and/or instantiate a service.

Another property of the reference architecture is that it makes no assumptions about how "thin" a VE is. It may take the form of an application, or a specialised service environment e.g. video on demand, or even a fully-fledged network architecture as proposed in [35], [36]. Finally, a VE may coincide with an implementation (EE instance) that is based only on one technology e.g. Java technology. In either case this is a design decision dictated by customer requirements and/or the VE owner.

Out of all the VEs residing in a node there must be a privileged one that is instantiated automatically when the node is booted up and serves as a controlled mechanism through which subsequent VEs may be created through the management plane. This privileged VE should be owned by the network provider who has access rights to instantiate the requested VE on behalf of a customer through a VE Manager (VEM). From this viewpoint the creation of VEs becomes a kind of meta-service.

2.1.2.2 Summary

ASP, ANTS and PLAN are existing execution environments in the Active Network Backbone [50]. ASP and ANTS offer in general a Java based Sub-OS for executing AAs. PLAN and SENCOM at EE level interpret script (carried in AA) to invoke a fixed function library. Finally the CANES approach calls plug-in modules in order to realise a generic processing function.

It is worth mentioning that almost all active and programmable network approaches that have been developed in several projects around the world, host also one or more EEs in their architecture. These EEs are approach-specific.

2.2 TRENDS AND EXPECTED EVOLUTION

It has been more than a decade since the introduction of programmability in the network elements (switches, routers etc) as the basis for the rapid deployment and customisation of new services. The next generation of heterogeneous networks are engineered to facilitate the integration and delivery of a variety of services.

Advances in programmable networks have been driven by a number of requirements that gave rise to a new business model, new business actors and roles [14], [7], [15]. More specifically, we are moving away from the “monolithic” approach whereby systems are vertically integrated to a component-based approach whereby systems are made of multiple components from different manufacturers, which may interact with each other through open interfaces to form a service [16]. The result is a truly open service platform representing a marketplace wherein services and service providers compete with each other, while customers may select and customise services according to their needs.

The problem space of programmable networks may well be represented by a two-dimensional model. Along the first dimension is the communication model that consists of packet header processing and forwarding, quality of service and congestion control mechanisms. Programmability along this dimension has been exercised by introducing service models like ATM or Diffserv [18] operating at the transport plane and then using the control plane to customise them resulting in different forwarding behaviours as perceived by the users.

In the second dimension, the computational model consists of “active” technologies whose origins can be traced to the areas of programming languages, object oriented and distributed programming, and operating systems. Recently, new hardware technologies like network processors [19], [20] have pushed the computational model even lower and closer to the physical interfaces of the networks elements. The computational model advocates higher amounts of computation and processing than the communication model as a means of pushing additional functionality inside the network to meet customer requirements. Programmability along this dimension is exerted by treating the network element (router, firewall, switch etc) as a programming environment, wherein service components may be deployed carrying out advanced.

Two distinct schools of thought, that address this problem space, can be identified. The first is represented by the Opensig community and was established through a series of international workshops [13], while the second one, Active Networks [12], is the result of a series of projects under the auspices of DARPA. Although these efforts seem from the outset to be antagonistic to each other and most of the time heavily dependent on the underlying networking technology and implementation, strong evidence points towards common features, that when put together as pieces of a puzzle they give rise to a picture that we believe is representative of the “programmable networks”. Recently such features have become the main focus of standardisation activities and in particular the IEEE P1520 [14] and the IETF ForCES protocol working group [22].

In this section, we will identify and elaborate on a number of such features arguing that they are the basic ingredients of the next generation NE architecture able to realise the yet elusive rapid service deployment. The ingredients identified are the Execution Environment, the building block approach and the principle of separation across the different operational planes necessary to support interoperability. Our analysis is based on the state of the art and is later used to explain the fundamentals of the FAIN project.

We also argue that because of strong evidence regarding the versatility of the building block approach, there is a specific type of EE that is going to be dominant in the next generation of Programmable Networks assisted by the widespread adoption of network processors. As a result we propose a new Network Element reference architecture that not only extensively uses this particular type of EE but also acts as a reference model to be used by service deployment mechanisms.

2.2.1 Open Signalling

- **The IEEE P1520**

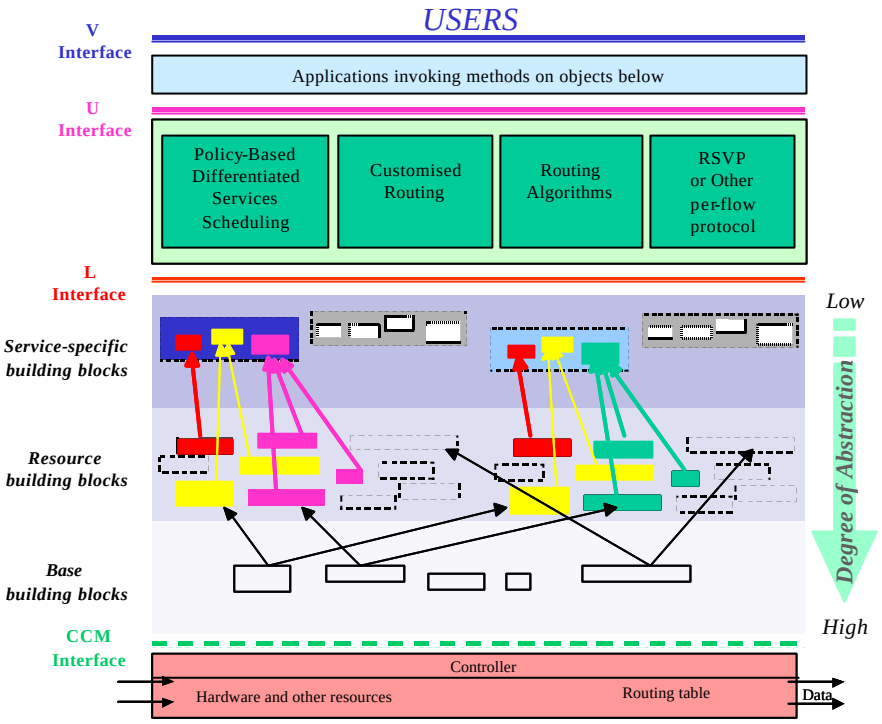


Figure 2.1. The P1520 Reference Model and the L-Interface abstraction model

The original motivation behind Opensig networks has been the observation that monolithic and complex control architectures may be restructured according to a minimal set of layers where the services residing in each layer are accessible through open interfaces thus providing the basis for service creation (composition). Eventually, a number of results out of the Opensig community were formalised by the IEEE Project 1520 standards initiative for programmable network interfaces and its corresponding reference model [14]. The IEEE P1520 Reference Model (RM) provides a general framework for mapping programming interfaces and operations of networks, over any given networking technology. Mapping diverse network architectures and their corresponding functionality to the P1520 RM is essential. The IEEE P1520 RM, depicted in Figure 2.1, defines the following four interfaces:

- *CCM interface*: The connection control and management interface is a collection of protocols that enable the exchange of state and control information at a very low level between the Network Element (NE) and an external agent.

- *L-interface*: It defines an Application Program Interface (API) that consists of methods for manipulating local network resources abstracted as objects. This abstraction isolates upper layers from hardware dependencies or other proprietary interfaces.
- *U-interface*: It mainly provides an API that deals with connection set-up issues. The U-interface isolates the diversity of connection set-up requests from the actual algorithms that implement them.
- *V-interface*: It provides a rich set of APIs to write highly customised software often in the form of value-added services.

CCM and L-interfaces fall under the category of NE interfaces, whereas U- and V-interfaces constitute network-wide interfaces.

Initial efforts through the ATM Sub-Working group (P1520.2), focused on telecommunication networks based on ATM and introduced programmability in the control plane [23]. Later, The IP Sub-working group extended these principles to IP networks and routers. Figure 2.1 also suggests a possible mapping of the P1520 RM to IP routers. However, their efforts focus on creating a generalised framework for designing interfaces not just for routers but also for any NE the core functionality of which is forwarding of traffic, e.g. switch, gateway etc [24]. For the remaining of this section we will focus on the activities of the IP working group as the most relevant to this document.

When the IP Sub-working group (P1520.3) first met, they faced two critical questions: a) which of the interfaces of the RM is the most important one in terms of maximising openness of the RM, and b) which is the right approach to achieve it.

Eventually, they decided that NE interfaces (CCM & L) are the most critical ones as they abstract the functionality and the resources found in the NE, thereby creating a kind of interoperability layer among different vendor's equipment and most importantly, allowing the requirements of network services residing in higher layers to be mapped in many different ways onto the capabilities of heterogeneous NEs.

The second question faced was the most complex one. Traditional packet and flow processing have long been the default network behaviour. However, with increasing intelligence being pushed into NEs, emerging devices will perform multiple functions, thereby defining a new class of network elements that extend behavioural functionality within the network transport. Thus, traditional routers and switches are going to be subsumed with next generation NEs capable of dynamically adapting to multi-function requirements. They are expected to include address translation, firewall enforcement, advanced flow differentiation, proxy activation, load balancing, and advanced monitoring.

Furthermore, in order to meet with the speed of the ever-rapid advancement in the technological frontiers, both in network devices and services/applications, an approach in specifying a standard should be based on a software architecture that allows extensive reusability of its modules. In other words, the development effort required to extend (i.e., proprietary or otherwise) the API should be minimised, so as not to hinder or delay deployment of emerging technological advances. The fundamental requirement levied on the standardisation process of the API, is that the standardisation itself shall not interfere with the future advancement and development of related technologies. In other words, it is required to be extensible. Furthermore, the extensible nature has to be available at all levels of abstraction within the API hierarchy. The standard API has to be extensible to accommodate new network devices, while at the same time be able to accommodate newly developed network services and applications. The former includes, for example, a proprietary hardware mechanism to accelerate a particular functionality, which could be realized by software in a "conventional" IP router.

Concluding, making a standard extensible for keeping up with the pace of innovation and differentiation you must make the composition mechanism part of the standard thus enabling seamless extensions of the API in the future.

Following this decision, P1520.3 also selected L-interface as their initial target for specification. In addition, the approach that was proposed to provide an answer to the second question is the building block approach [25], [26], which consists of three layers of abstraction that define a model for specifying the API (Figure 2.1).

The model enables network device programmability from two complementary perspectives, corresponding to an association with the layers of the L abstraction model, primarily service-specific and resource. This allows, for example, upper level interfaces to create or program completely new network services using generic resource abstractions or to modify existing services using service-specific abstractions, which are themselves built on generic resource abstractions. The third layer is introduced to facilitate common device programmability by means of composition, via a standard set of base building block abstractions, on which both the service-specific and resource layers are built.

More specifically, the upper part of the L interface is the service-specific abstraction layer of the NE. The Service-Specific Building Block (SSBB) abstractions at this layer expose "sub"-interfaces associated with underlying behaviours or functions, state or policies on the local node that have concrete meaning within the context of a particular supported service (e.g. Differentiated Services). The idea here is that an administrator or Internet Service Vendor (ISV) need only program the device within the context of the service (i.e., preferably industry standardised), rather than deal with low-level abstractions associated with fundamental resources (e.g. scheduler, dropper) of the network device. Thus, to deliver the required service-specific behaviour, he or she needs only modify, update or provision the service abstraction at the level that they understand or have a need (or privilege) to supplement.

Alternatively, the middle part of the L interface abstraction model is the resource abstraction layer of the NE. The abstractions here are termed Resource Building Blocks (RBB), from which primitive behaviours (e.g., Diffserv PHB [18]) or new behaviours can be built. We envision the programmer is a sophisticated ISV developer or network software architect, who is aware of underlying resource abstractions (not implementation) of a NE (e.g. router), and can construct new behaviours or functions, change state or policies within the context of the generic abstraction, without specific knowledge of the underlying vendor device implementation.

The maximum degree of abstraction is achieved at the lowest layer of the abstraction model. It is at this layer that the composition mechanism is abstracted and becomes part of the standard. The idea behind the Base Building Blocks (BBB) is to have abstractions that have no service or resource significance from a NE behavioural or packet processing perspective. These base blocks serve the needs of the programmer, only in an inheritance fashion such that the abstractions above the base layer (namely resource or service-specific) can be designed appropriately to create new functional service behaviours or resources or modify (enhance) existing ones in a consistent, standard object-oriented manner.

As a result of the approach a number of APIs are defined at each layer with the ones at the BBB layer providing methods that allow RBBs to be composed in such a way that they form SSBBs constructs. In this way, using the APIs of the BBB layers and defining as RBBs components like classifier, meter, shaper, queue and scheduler, one can create a Diffserv SSBB the API of which will be the collection of the APIs of the individual RBBs. To this end, by standardising a small set of RBBs one can create any SSBB that is required by specific network services. Alternatively, new RBBs may be introduced by means of inheritance from the BBB layer and deployed in the NE in order to support new network service requirements or enhance/extend existing functionality.

- **The IETF ForCES**

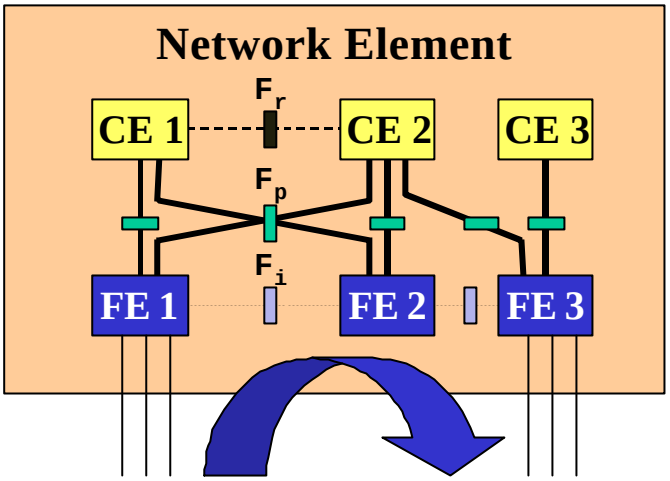


Figure 2.2. ForCES Architectural Representation of NE

The Opensig community have long advocated the benefits of a clear distinction between control and transport plane. Recently, a working group of IETF, called ForCES (Forwarding and Control Element Separation) was formed with a similar objective to that of P1520, namely, "by defining a set of standard mechanisms for control and forwarding separation, ForCES will enable rapid innovation in both the control and forwarding planes. A standard separation mechanism allows the control and forwarding planes to innovate in parallel while maintaining interoperability" [22], [27].

According to [26], the NE is a collection of components of two types: control elements (CE) and forwarding elements (FE) operating in the control and forwarding (transport) plane, respectively. CEs host control functionality like routing and signalling protocols, whereas FEs perform operations on packets, like header processing, metering, scheduling etc when passing through them. CEs and FEs may be interconnected with each other in every possible combination (CE-CE, CE-FE and FE-FE) thus forming arbitrary types of logical topologies (see Figure 2.2). Every distinct combination defines a reference point, namely, Fr, Fp and Fi. Each one of these reference points may define a protocol or a collection thereof, but ForCES protocol is only defined for the Fp reference point.

However, FEs do not represent the smallest degree of granularity of the NE functionality. Furthermore, as they implement the ForCES protocol they must facilitate CEs to control them in terms of abstracting their capabilities, which, in turn may be accessed by the CEs. It is at this point that the ForCES group faced a similar challenge as the IP working group in P1520 which they formulated it as follows: Since FEs may manifest varying functionality in participating in the ForCES NE, "the implication is that CEs can make only minimal assumptions about the functionality provided by its FEs" [28]. As a result, CEs must first discover the capabilities of the FEs before they can actually control them.

The solution they suggest is captured in the form of an FE Model [28], while two of its requirements that must satisfy pertain to the problem of an extensible standard. The first mandates that the FE model should provide the means to describe existing, new or vendor specific logical functions found in the FEs, while the latter demands to describe the order in which these logical functions are applied in the FE [15].

In the ForCES FE model, they use a similar approach to the building block approach of the P1520.3 working group, by encapsulating distinct logical functions by means of an entity called, FE block. When this FE block is treated outside the context of a logical function, it becomes equivalent of the base building blocks. When someone looks what is inside every FE block then it becomes a resource building block. Similarly, FE blocks eventually are expected to form an FE block library – in principle extensible-, which will be part of the standard and the basis for creating complex NE behaviours, although dynamic extensions thereof may be possible. Of course there are differences between the two initiatives but the main ideas are very close so we expect that in the future they will fully converge.

A type of model like the FE model is useful when CEs attempt to configure and control FEs. ForCES has identified three levels of control and configuration, namely, static FE, dynamic FE, and dynamic extensible FE control and configuration. The first assumes that the structure of the FE is already known and fixed, the second one allows the CE to discover and configure the structure of the FE although selecting from a fixed FE block library, whereas the third one is the most powerful that allows CEs to download additional functionality, namely FE blocks, onto FEs at runtime. Currently ForCES is mainly, focusing on the first level of control and configuration.

2.2.2 Active Networks (DARPA)

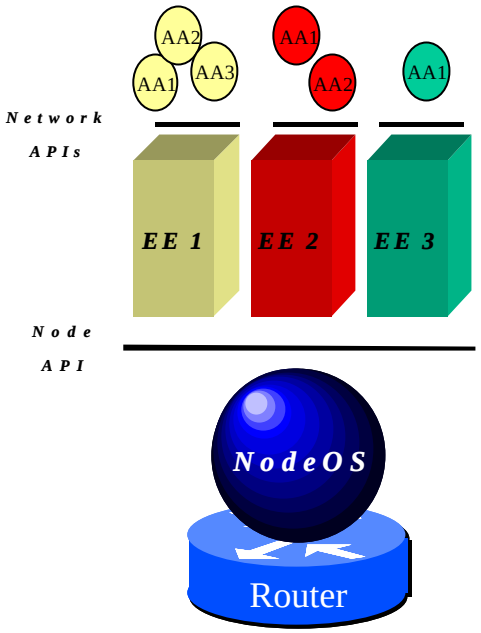


Figure 2.3. The Active Node Architecture

Active Networks transform the store-and-forward network into store-compute-and-forward network. The innovation here is that packets are no longer passive but rather active in the sense that they carry executable code together with their data payload. This code is dispatched and executed at designated (active) nodes performing operations on the packet data as well as changing the current state of the node to be found by the packets that follow. In this context, two approaches can be identified based on whether programs and data are carried discretely, namely within separate packets (out-of-band) or in an integrated manner, i.e. in-band.

In the discrete case, the job of injecting code into the node and the job of processing packets are separated. The user or network operator first injects his customised code into the routers along a path. Then the data packet arrives, its header is examined and the appropriate pre-installed code is loaded to operate on its contents [4], [76]. Separate mechanisms for loading and executing may be required for the control thereof. This separation enables network operators to dynamically download code to extend node's capabilities, which in turn they become available to customers through execution.

At the other extreme lies the integrated approach where code and data are carried by the same packet [2]. In this context, when a packet arrives at a node, code and data are separated, and the code is loaded to operate on the packet's data or change the state of the node. A hybrid approach has also been proposed [29].

Active networks have also proposed their own reference architecture model [11] depicted in Figure 2.3. According to it an Active Network is a mixture of active and legacy (non-active) nodes. The active nodes run the node operating system (NodeOS) –not necessarily the same- while a number of Execution Environments (EE) coexist at the same node. Finally a number of active applications (AA) make use of services offered by the EEs.

The NodeOS undertakes the task of simultaneously supporting multiple EEs. Accordingly, its major functionality is to provide isolation among EEs through resource allocation and control mechanisms, and providing security mechanisms to protect EEs from each other. It may also provide other basic facilities like caching or code distribution that EEs may use to build higher abstractions to be presented to their AAs. All these capabilities are encapsulated by the Node interface through which EEs interact with the NodeOS. This is the minimal fixed point at which interoperability is achieved [30].

In contrast EEs implement a very broad definition of a Network API ranging from programming languages to virtual machines like the Spanner VM in Smart Packets and bytecodes, to static APIs in the form of a simple list of fixed-size parameters etc [31]. To this end, EE takes the form of a middleware toolkit for creating, composing and deploying services.

Finally, the AN reference architecture [11] is designed for simultaneously supporting a multiplicity of EEs at a node. Furthermore, only EEs of the same type are allowed to communicate with each other, whereas EEs of different type are kept isolated from each other.

2.2.3 Key Concepts

The analysis of the state-of-the-art presented in this chapter aims to identify those ingredients that may serve as building materials and principles to the next generation NE architecture. In other words, we are trying to address the question, how may programmable networks be evolved? We aspire that the results produced in FAIN provide a satisfactory answer.

First and foremost, we consider the concept of EE as the basis of the next generation NE architecture that greatly facilitates the definition of a reference architecture. Such architecture acts as a reference point that service deployment algorithms need in order to make decisions about where service components can be deployed, which is the appropriate implementation technology of these components, how the deployed components are linked with existing ones that are running in the NE etc.

But what exactly is an EE, what elements is it comprised from and are these elements part of the architecture or part of its chosen implementation? Furthermore, is it possible to identify specific types of EEs that are implementation independent? Scanning the literature we can trace a variety of answers regarding the exact characteristics of an EE.

Conceptually an EE is the active network's programming environment [32] when instantiated it becomes the runtime environment of a process or a process itself [33]. This programming environment may be centred on a particular language and may export some API that encompasses elements like a Java Virtual Machine [32], [11], toolkits used for building

AAs (services) [4], [33] or even interfaces to access generic services that AAs may customise building value added services. EEs have also been proposed as extensions of the NodeOS for those that are allowed to be extensible [30]. The latter has an impact on where to draw the boundary between EE and NodeOS known as the Node interface.

The fact that the AN reference architecture [11] simultaneously supports multiple EEs, implies that EEs are also treated as principals based on which authentication, authorisation and resource control takes places. Services and users that use an EE are represented by this principal, which is the only valid entity allowed to access NodeOS facilities. To this end, the EE concept is overloaded with the characteristics of a virtual environment. Prototypes proposed in [15], [34] may be interpreted in this way. Finally, EEs have been characterised not by the choice of technologies but rather by the services they offer and the architectural plane they operate at, namely, control, management, and transport [35], [36].

It is evident that in the majority of cases the boundaries between architecture and implementation are blurred that, in turn, makes it is very difficult to come up with a clear definition of an EE. Lack of an unambiguous definition impedes any effort to propose a reference NE architecture that not only does it encompass most of the research efforts so far, but also it is instrumental in designing a middleware for service creation and deployment. This has been the subject of one of the research activities in FAIN.

The second of these ingredients is the right approach based on which EEs must be designed. As it has been argued, the approach must satisfy the requirements for composability, extensibility, and vendor independence. We believe that the building block approach is the right one for designing EEs. Recently, a new research activity has been reported in [37], which uses a similar approach applied to redesign of protocols that do not imply a layered IP architecture.

The final ingredient mainly deals with the problem of interoperability and the NE itself. It comes in the form of the separation principle among the different operational planes and the abstraction of the functionality at each one of the planes by means of open interfaces.

In section 4 the FAIN architecture is introduced we describe how combining these ingredients programmable networks may evolve.

3 REVIEW OF MANAGEMENT SYSTEMS TECHNOLOGIES

The research activities described in the previous chapter have clearly focused on the synergy of three concepts: network virtualisation, open interfaces and platforms, and intelligence in the network. The network is no longer an entity providing basic connectivity but rather is treated as a service-enabling platform, which is open, intelligent and offers a variety of virtualised facilities under the authority of different communities. Consequently, the management of such network has to take into account a whole new range of problems and complexity. To this end, it must support co-existence of different management strategies thus facilitating customisation, interoperate with different vendors' equipment, and dynamic extensibility of its functionality to support new services and their deployment.

In this chapter we analyse different research projects developed in the area of management of active and programmable networks, which try to encompass the whole range of solutions proposed. More specifically we have divided the analysis in two big areas composing management: network management (i.e. we focus in policy-based management of active networks) and active service provisioning. At the end of the chapter we elaborate around the direction that current research and standardisation activities seem to take for the short future.

3.1 STATE OF THE ART

In this section we analyse the current state of the art in management of active and programmable networks. More specifically, we have divided the section in two parts: the first one dealing with network and element management of active and programmable networks, and the second on dealing with the state of the art of active service provisioning research.

3.1.1 Network and Element Management

Active networks and particularly, management of active networks, is still a young research field. In consequence, no standardisation activities have been started for management of active networks. On the other hand, Policy-based management is an emerging technology for the management of telecommunications networks that can be adapted, as we have demonstrated within FAIN and a number of other projects briefly described afterwards, to deal with Active Networks. More specifically, policy-networking technology eases the handling of Active Networks specificities. For example, policies are particularly suited for delegating management responsibility, essential to enable the customisability of network resources. Also, the device-independent property of policies is optimum for the management of heterogeneous network technologies. Finally, policies permit a more automated and distributed approach to management, taking decisions based on locally available information according to a set of rules.

Both the Internet Engineering Task Force (IETF) [51] and the Distributed Management Task Force (DMTF) [52] 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 [53]. The IETF is also working in that field, in co-operation with DMTF [54], 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 [55].

Aside from the standardisation activities many research projects have covered the field of policy-based management. Among these, the more relevant ones for FAIN are: the Ponder and Jasmin projects. The Ponder project has been one of the first technology and manufacturer-independent policy-based management frameworks. Aside, it defines a policy specification language where from many concepts have been re-used in FAIN. The Jasmin project, although not being an Active Networks-related project it explores the automation and distribution of policies and policy-decisions, properties of the highest relevance also in FAIN.

When focusing at policy-based management of active networks we realize that up to now there are not many efforts that analyse the synergies that can be obtained from joining Active and Policy-driven Networking technologies. Some of the more widely known and accepted ones are: Seraphim, ANDROID, PxP, A-PBM, Policy networking using active networks and Policy specification for programmable networks.

- **Ponder**

The Ponder project [5] has had a good acceptance within the research community and its results have been used in many research projects using policy-based management. Ponder defines a language and framework for specifying security policies that map onto various access control implementation mechanisms for firewalls, operating systems, databases and Java. It supports obligation policies that are event triggered condition-action rules for policy based management of networks and distributed systems. Ponder can also be used for security management activities such as registration of users or logging and auditing events for dealing with access to critical resources or security violations. Key concepts of the language include roles to group policies relating to a position in an organisation, relationships to define interactions between roles and management structures to define a configuration of roles and relationships pertaining to an organisational unit such as a department.

- **Jasmin**

The Jasmin project [57] aims to evaluate, enhance and implement the distribution and invocation of network management scripts with distributed network management applications. The implementation supports multiple languages and run-time systems. As part of the Jasmin project as set of classes have been added to support policy-based configuration management of Linux Diffserv nodes. In particular, general policy management language extensions, domain specific policy management language extensions and drivers realising the mappings between domain specific policies and the underlying device-level mechanisms, have been realised.

- **Seraphim**

One of the first projects to work with policies in active networks was the Seraphim project [58]. It 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.

- **ANDROID**

The ANDROID (Active Network DistRibuted Open Infrastructure Development) project [59], proposes a policy and event driven architecture for the management of Application Layer Active Networking (ALAN) [60] networks. The project is mainly focused to the management of active servers, where programmability up to the application level is allowed. Nonetheless, they also consider a reduced management of the active routers, i.e. configuration of routes to users towards their assigned active server.

The ANDROID management framework is policy-based and event-driven. Policies and events are introduced into a new XML document called Notification. When a user wants to install a new service inside an active server, it sends an event to the network operator. The operator initiates the resources and security checks, based on available policies, and then loads the active service. Active services are continuously monitored, so that if an unexpected behaviour is detected corrective policies can be enforced to correct this behaviour.

- **Policy eXtension by Policy (PxP)**

In [61] a method for the dynamic extension of a policy-based management system by means of policies in active networks is suggested. The Policy Extension by Policy (PxP) proposal is limited to the extension method, so it must be included within another management framework. The method defines two types of policies for realising this extension, i.e. Policy Definition (PD) policies and Policy Extension (PE) policies. On the one hand, PD policies allow a user to add a new type of policy into the Policy Server specifying the correct syntax and restrictions. Then, through PE policies users can specify the corresponding methods for translating the new policies types into commands on different types of network nodes. Both PD and PE policies are defined either by network operators or an application.

- **Active Policy-Based Management (A-PBM)**

In [62] a framework to allow the interoperability between different ISP management domains satisfying end-to-end requirements given by users is proposed. The proposed framework extends the policy-based management framework proposed by the IETF by including capsules for the communication between the different components of this framework. Capsules represent user requirements and are used for service negotiation and network elements configuration. They have defined three types of capsules: one to request decisions from the PEP to the PDP, one for notify decisions from the PDP to the PEP and a third one to negotiate between ISPs.

- **Policy networking using active networks**

In [63] a management framework designed for reducing management traffic by allowing network elements to take decisions is proposed. This is done by defining active packets, that might even contain a portion of a policy decision point, which are executed inside network elements; thus, allowing network elements to take autonomous, intelligent decisions.

- **Policy specification for programmable networks**

Finally, in [64] the research group responsible for the Ponder project analyses and suggests a notion and framework for specifying policies related to programmable networks. The proposal is mainly based on Ponder concepts such as policy-grouping based on their roles.

3.1.2 Active Service Provisioning

In this section, we discuss existing active and passive network systems from a service deployment perspective. The systems were chosen in a way to get a sampling of different design decisions.

Service deployment in a network can be subdivided into two levels:

- The network level, where components have knowledge of the network topology and of the capabilities of the active nodes to be able to choose which active nodes to deploy the software onto.
- The node level, where software must be really deployed within the node environment.

A comprehensive framework must support service deployment at both levels. Due to the loose coupling between the two levels, the design of service deployment mechanisms at each level can be tackled to a great extent independently. The design space is similar for both levels [74].

A first decision designers of service deployment systems have to take is between in-band and out-of-band service deployment. In-band deployment refers to a system, where the service logic is distributed in the same path as payload data. In-band deployment often refers to a broad code deployment onto a lot of active nodes (nodes in the deployment path); so it may be useful to deploy a given service all over an architecture and in this sense may be compared to a CDN (Content Delivery Network) [75] architecture where the service is deployed onto all the nodes of this architecture. Out-of-band deployment, on the other hand, refers to an architecture where service deployment and payload data use logically and/or physically distinct communication channels. That is, service deployment information is exchanged via the control or management plane. Out-of-band deployment focuses more on deployment onto specifics targeted active nodes: the service is deployed onto given nodes depending on their locations (edge routers for instance) or the network topology, depending on the nodes capabilities (type of environment execution, current CPU load, available bandwidth), etc.

Considering the history of active networks, in-band approach has been the first deployment method designed and implemented (ANTS active node for instance) but we may argue, that now, the out-of-band approach is more and more used and is going to be the leading way to deploy code in the new active platforms, mainly because of the security considerations.

In the in-band approach the deployment request (or the reference of a service) is carried inside an active packet but the code may be retrieved from a code repository using an out-of-band approach and not always conveyed in the data payloads. Some active platforms using this approach are ANN [29] (Active Network Node) or ANTS [4] (Active Node Transfer System) or PLANet/SwitchWare [76].

The out-of-band approach is a bit different as it assumes that a “network entity” is used to deploy the service onto active nodes. This network entity may be a service provider itself, a network operator or a component aiming at the service deployment but operating at the network level. Whereas in the two first cases, the code is initiated to a specific node (already chosen), the last case (network deployment component) is more generic since the network component may have a view of the network topology, may know the status of the active nodes (CPU load, available bandwidth…), the latter might then deploy the service on given nodes depending on these capabilities. For instance, the ALAN [77] platform can be classified in the first category because the actor in charge of the deployment must know onto which active node to deploy the service to. On the contrary, the FAIN platform might be classified in the second category because a network deployment component is used [78]. The latter is in charge of choosing the appropriate active nodes depending on constraints like the topology, the load of the nodes.

A second dimension of the service deployment design space distinguishes between centralised and distributed service deployment mechanisms. In this context, the design choice refers to the method of deployment information processing.

We observe that active networks typically use a distributed, in-band approach at the network level. Greater variety can be found in the approaches for the node level. The chosen approach depends on targeted services, as well as performance and security considerations.

The following sections discuss the approach to service deployment adopted by existing active network systems.

- **ANN**

The *Active Network Node (ANN)* architecture makes use of active packets to deploy services. These packets feature a reference to a *router plug-in*, which contains the service logic. If not cached locally on a node, router plug-ins are fetched from a code server and installed on the node.

Using active packets, the service deployment mechanism of this system can be classified as a *distributed, in-band* approach at the network level. It is distributed because the service logic is installed on active network nodes traversed by active packets. Whether an active network node is traversed or not depends on the forwarding tables in the nodes, which are set up by distributed routing algorithms. It is in-band because necessary information is contained in active packets, which also carry payload data.

At the node level, an out-of-band mechanism is used. That is, router plug-ins are fetched from a code server, using a different logical communication channel. As plug-ins may modify all fields of active packets, the choice between a centralised or distributed approach is left to the service designer.

- **ANTS**

From a service deployment viewpoint, ANTS is similar to ANN. It also uses active packets, which contain references to *code groups*, i.e. the service logic.

At the network level, the same comments as for ANN apply.

At the node level, however, it is interesting to note that while using an out-of-band approach, ANTS efficiently mimics an in-band deployment mechanism. That is, if an active packet arrives at a node, service logic is, if not cached locally, retrieved from the cache of the previously visited node. Therefore, active packets and service logic generally follow the same path. The out-of-band approach, however, allows for an efficient use of network bandwidth because the service logic follows the first active packet of a stream. Subsequent active packets of the same stream will make use of the cached service logic. Therefore it is not necessary to transmit the service logic with each active packet. As in the case of ANN, the choice between a centralised and a distributed approach is left to the service designer.

- **PLANet/SwitchWare**

SwitchWare is an architecture that combines active packets with *active extensions* to define networked services. Active packets contain code that may call functions provided by active extensions. Active extensions may be dynamically loaded onto the active node. *PLANet* implements this architecture using a safe, resource-bounded scripting language (called PLAN) in the active packets and a more expressive language to implement active extensions.

At the network level, service deployment is implemented in a *distributed, in-band* way, using active packets similar to ANTS and ANN.

An interesting service deployment characteristic of the SwitchWare architecture is found at the node level. In fact, both in-band and out-of-band service deployment is used to combine the advantages of both worlds. Active packets contain code (in-band) that can be used like a glue to combine services offered by dynamically deployed active extensions (out-of-band). Similar to ANN and ANTS, the content of active packets may be modified by active extensions. Therefore, the choice between a centralised and distributed way of deployment is left to the service designer.

- **HIGCS**

HIGCS uses a *distributed, out-of-band* approach to service deployment at the network level. Similar to hierarchical routing schemes, nodes build clusters and elect a cluster leader to aggregate information (e.g. node capabilities) and to represent it to the upper hierarchy level. The information exchange is based on a specific control protocol (e.g. an extension to a hierarchical routing protocol) and therefore an out-of-band approach. It is distributed because cluster leaders process (aggregate, distribute) information relevant to service deployment within their cluster.

As HIGCS is intended for the network level deployment, a discussion of the node level service deployment is not applicable. Furthermore, HIGCS is targeted towards programmable networks, and, as a consequence, does not deal with code deployment.

3.2 TRENDS AND EXPECTED EVOLUTION

3.2.1 Element and Network Management

In our context, network management [65] means deploying and coordinating resources in order to administer and operate Active Networks, with the objective of achieving the required quality of service (QoS), thus fulfilling the expectations of both the owners and the users of the network [66]. Methods to predict [67] or rapidly detect failures [68] and alert the relevant personnel to take remedial action can substantially reduce user inconvenience.

The general trend in network management is to achieve scalability in functionality. The research community constantly comes up with novel ideas for optimising efficiency and functionalities, only to fall short of having global end-to-end management capabilities. SNMP is still the de facto management protocol on the Internet.

In recent years, new management paradigm proposals tried to overcome some of the key deficiencies of SNMP. The Management by Delegation (MbD) [69] paradigm proposes a distributed hierarchy of managers that solves the problem of polling distance between the manager and the agent. MbD was expected to be a scalable proposition when compared to the SNMP model because if data analysis is only conducted at the management station (as is the case for the latter), it will require data access and processing rates that do not scale up for large and complex networks (e.g., the Internet)

While the MbD approach is a trend away from centralised approach, i.e., pushing intelligence from management system to managed element (using mobile agents for code mobility), the policy-based approach is a trend towards simplification of configuration by means of high-level rules.

The introduction of automation of management tasks involves the most significant change with respect to current implementations of management tools with existing technologies (e.g., SNMP). The mobile agent [70] and Active Networking [71] technologies have been extensively investigated over the last several years for this primary interest. The programmable networking paradigm offers the possibility of utilising dedicated plug-ins for per-flow monitoring.

Automation in the network environment [72] has been proposed many times during the past 15 or so years, for example, in routing at switches, and it is arguable that a large-scale adoption and implementation is around the corner. However, operators have been nervous about adopting extensive automation.

To cope with interoperability, middleware technologies like CORBA and Java RMI are used for inter-domain management [73].

3.2.2 Active Service Provisioning

There a number of standardisation activities going on that might have an impact on the service deployment for active networks field. Among these, the OSGI [79] (Open Service Gateway Initiative), a bit different from active networks, may be a source of inspiration for the active network deployment [80]. Even though OSGI focuses on gateways, like the ones used in the home networks (to manage the home networks and to interact with the broadband network as well), the mechanism they have designed is component-based. Indeed a service is seen as a set of components, offering interfaces and the deployment is done using this “bundle”.

Another interesting standardisation activity is the OMG [81] (Object Management Group), standardising the CORBA technology, has designed the CCM [82] (CORBA Component model) that describes the definition of a component and the interfaces it offers to others. We may imagine that the component-based deployment is the next step in the deployment process for active networks as well and that it can find ideas in the previous ones.

Network operators consider active packet-based networks to be unsafe and insecure, because they have not sufficient control over the code to be deployed. Therefore, a component-based approach, as proposed by FAIN and others, is necessary to convince operators of the technology [83]. This opens a field for **standardisation of component behaviour.** That is, component names should have well-defined semantics. In the FAIN approach, information has been defined in XML to specify the characteristics of service (type of execution environment, code location, service provider, network topology constraints, etc.).

4 FAIN ENTERPRISE MODEL

In this section, be introduce the FAIN enterprise model. From an abstract point of view, an enterprise model consists of (1) actors, i.e., persons, companies, or organisations of any kind that own, use, support and/or administrate parts of an (active) networking system[1], (2) the various (business) roles these actors play, and (3) the contractual relationships between these roles, i.e., the reference points. The term "business role" refers to the specific behaviour, properties, and responsibilities assigned to an actor with respect to an identifiable part of the networking system and the related activities that are necessary to keep this part of the system operational. The set of (FAIN) actors is mapped onto the set of business roles occurring in the FAIN context, i.e., each role is understood to be played by at least one actor which may, however, perform more than one role. The relationships between roles are expressed as reference points which determine the interactions (interfaces and action sequences, together with related properties and constraints, granted rights, etc.) between the actors in the respective roles.

The FAIN actors are the "usual suspects": (a) the providers, i.e., companies operating networks (by managing the respective hardware and software resources) and offering services over the network (employing their own resources or those of the – traditional – network operators), etc.; (b) the users, i.e., individuals or companies which use the proffered connectivity and services; and (c) the manufacturing companies which support the providers by developing (and vending) the necessary hardware and software components. The manufacturers are not in the main focus of the FAIN context, even if their role is an important one and, as such, briefly reflected in the enterprise model, in order to make the picture complete.

The FAIN enterprise model only describes the business roles which may be taken by the above listed actor companies (and obviously, each actor company may act in more than one role if this suits its business). The purpose of defining business roles is to partition an active networking system in a way that responsibilities are identified, such that they can be clearly separated and assigned to different actors. Consequently, the enterprise model has to determine the interactions that are expected to take place between the actors acting in the respective business roles, in order to allow for independence and "compositionallity" of the partial systems provided by the actors. The reference points are the means to describe these interactions which comprise the administrative relations, i.e., how to establish/handle the involved business relations and sometimes even legal issues (the "business model" in the terminology of the Network Management Forum NMF) and the technically necessary interaction schemes that have to be supported by the respective interacting components, their functionality and their capabilities, the qualities they have to guarantee, etc., in short, the necessary technical (interface related) contracts that have to be established between the components in the realm of the various actors in their respective roles.

4.1 ROLES

The business model for active network/service provisioning is depicted in Figure 4-1. The highlighted/grey-shaded business roles are considered to be within the focus of the FAIN project, the other roles in the figure are not further detailed beyond the brief description given in the text that follows. The FAIN business model is composed of the following roles:

[1] The active networking system is the "FAIN universe of dis course", i.e., the (active) network (router and their software) together with all the services provided in relation with it.

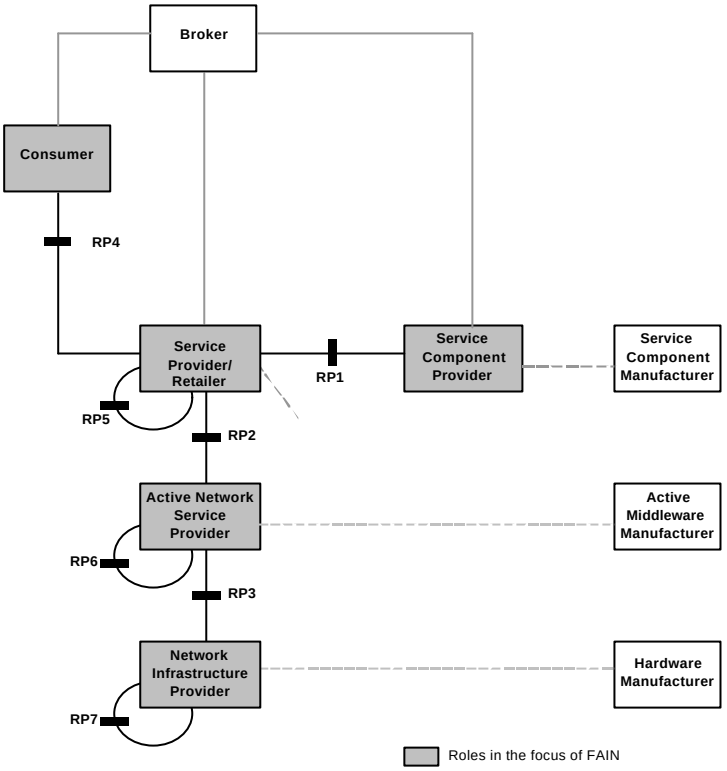


Figure 4-1: The FAIN Business Model

Consumer (C): The Consumer is the end user of the active services offered by a Service Provider. In FAIN, a Consumer may be located at the edge of the information service infrastructure (i.e., be a classical end user) or it may be an internet application, a connection management system, etc. In order to find the required service among those proffered by the Service Providers the Consumer may contact a Broker and, e.g., use its service discovery features.

Service Provider (SP): A Service Provider composes services, including active components delivered by a Service Component Provider, deploys these components in the network via the ANSP, and offers the resulting service to the Consumers. The services provided by a Service Provider may be end user services or communication services. A Service Provider may federate with other Service Providers in order to build more complex services. Descriptions of the offered services are published via the Broker service.

Active Network Service Provider (ANSP): An Active Network Service Provider provides facilities for the deployment and operation of active components into the network. It provides Virtual Environment facilities, (one or more) for active service components to Service Providers. Together with the Network Infrastructure Provider, it forms the communication infrastructure

Network Infrastructure Provider (NIP): A Network Infrastructure Provider provides managed network resources (bandwidth, memory and processing power) to Active Network Service Providers. It offers a network platform including AN components to the Active Network Service Providers who can build their own Execution Environments, and proffers basic IP Connectivity[2] which may be based upon traditional transmission technology as well as emerging ones (both wired and wireless)[3].

[2] Since FAIN focusses on Internet technology, a basic network infrastructure consists on the following characteristics: presence of a physical network; IPv4 and/or IPv6 forwarding mechanism; default IP routing protocol; if RP 7 is implemented: peering service to other NIPs or ISPs; element management interface.

[3] The Service offered by the NIP may also be used by current Internet Providers.

Service Component Provider (SCP): A Service Component Provider is a repository of active code that may be used by the Consumer and the Service Provider. A Service Component Provider may publish its components via a broker.

Service Component Manufacturer (SCM): A Service Component Manufacturer builds service components and active code for active applications and offers them to Service Providers, Consumers and Service Component Providers in appropriate form (e.g., as binary or as source code). The Service Component Provider is not a genuine FAIN business role since it is not directly involved in the main FAIN business process.

Middleware Manufacturer (MM): An Active Middleware Manufacturer provides platforms for active services (including the Execution Environments) to the Active Network Service Provider. The Active Middleware Manufacturer is not a genuine FAIN business role since it is not directly involved in the main FAIN business process.

Hardware Manufacturer (HM): A Hardware Manufacturer produces programmable networking hardware for the Network Infrastructure Provider. The Hardware Manufacturer is not a genuine FAIN business role and will not be detailed further in the FAIN Enterprise Model.

Retailer (RET): The Retailer (according to the TSAS model) acts as "access point" to the services provided by Service Providers to Customers; it has to be capable of negotiating and managing the end user contract (handling authentication and security issues as well as accounting and billing, etc.) and it has to allow the service life cycle management by the user (subscription and customisation of a service, administrative interactions, etc.).

Broker (BR): The Broker collects, stores and distributes information about (1) available services, (i.e., those offered by the Service Providers), (2) the administrative domains in the active networking system, and (3) service components (i.e., code offered by the Service Component Providers). The FAIN broker basically resembles the "Broker" role in the TINA business model. The broker is no genuine FAIN business role and will not be detailed further in the FAIN Enterprise Model.

4.2 REFERENCE POINTS

Among the business roles, the following reference points are defined:

RP1 (SCP-SP): Contracting, subscription and transfer of components in a client server situation; Information about Execution Environments should be exchanged to determine which components to transfer

RP2 (ANSP-SP): SP delivers components to be injected in the network by the ANSP; SP requests generic network services, processing resources and code injection from the ANSP;ANSP allocates resources depending on availability; ANSP provides information about capabilities of network services; SP is responsible for service session management, according to the agreed service level (ANSP-SP);ANSP is responsible for network service management, according to the agreed service level (ANSP-SP)

RP3 (NIP-ANSP): NIP provides and allocates physical active networking resources; ANSP requests resources (network, processing, transmission, etc.) from the NIP in order to provide network services and is responsible for injection of active code.

RP4 (SP-C): The Consumer requests and uses services offered by the Service Provider. The SP provides the service with its functionality. It also subsumes the role of the Retailer as service access point (i.e., it determines the service contract).

In addition, RP5 (SP-SP), RP6 (ANSP-ANSP) and RP7 (NIP-NIP) deal with federation among roles of the same kind. For instance, a Service Provider may provide features of the services of other co-operating Service Providers and it may use these Service Providers to offer facilities of different Service Component Providers it is not associated with directly.

5 FAIN REFERENCE ARCHITECTURL MODEL

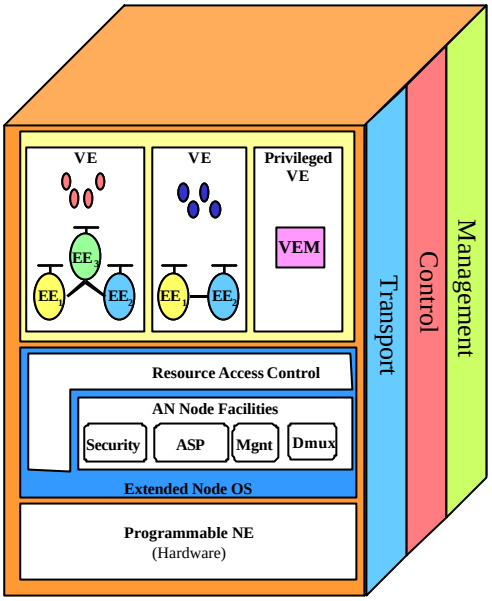


Figure 5-1: The FAIN NE Reference Architecture

The FAIN NE reference architecture depicted in Figure 5-1 describes how the ingredients identified in Section 4. can be combined synergistically to build next generation NEs capable of seamlessly incorporating new functionality or dynamically configured to change their behaviour according to new service requirements.

One of the key concepts defined by the FAIN architecture is the EE. In FAIN, drawing from an analogy based on the concepts of class and object in object-oriented systems, we distinguish EEs between the EE type and the EE instances thereof. An EE type is characterised by the programming methodology and the programming environment that is created as a result of the methodology used. The EE type is free of any implementation details. In contrast, an EE instance represents the realisation of the EE type in the form of a runtime environment by using specific implementation technology e.g. programming language and binding mechanisms to maintain operation of the runtime environment. Accordingly, any particular EE type may have multiple instances while each instance may be based on different implementations. Such distinction allowed us to address the issue of the principles that must govern and the properties that must be possessed by next generation NEs, from the issue of how to build such systems.

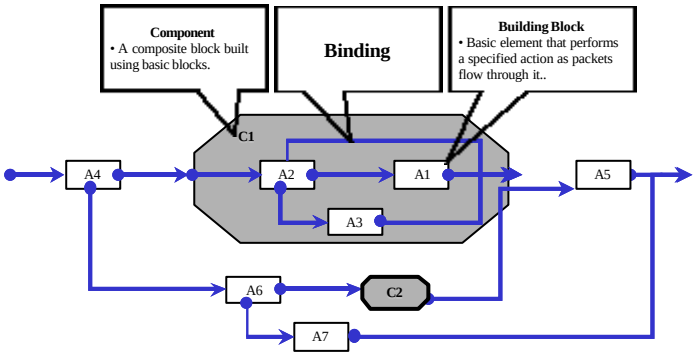


Figure 5-2: EE Type: The Programming Environment

The programming methodology that was used as part of the FAIN EE type was the building block approach according to which services break down into primitive, distinct blocks of functionality, which then may be bound together in meaningful constructs. To this end, services can be rebuilt from these primitive forms of functionality, i.e. the building blocks, while building blocks may be reused and combined together in a series of different arrangements as this is dictated by the service itself. The result of this process is the creation of a programming environment like the one depicted in Figure 5-2. In FAIN we have built two different EE instances, a Java EE and a Linux kernel-based EE, of this particular EE type [18].

FAIN architecture also allows EEs to reside in any of the three operational planes namely, transport, control and management while they may interact and communicate with each other either across the planes or within a single plane. In fact, it is not the EEs that communicate but rather distributed service components hosted by them part of deployed network services which can be accessed by applications or higher level services by means of the network API they export. EEs (instances) are the place where services are deployed. Services may well be extensible in the sense that the programming methodology and the corresponding environment (EE type) support service extension while they can access services offered by other EEs to achieve their objectives and meet customer demands. For example a service uses the code distribution mechanism to download its code extensions. The extension API then becomes part of the overall service interface.

Furthermore, FAIN separates the concept of the EE from that of the Virtual Environment (VE). We argue that the concept of an EE as defined previously and that of a VE are orthogonal to each other. In fact a VE is an abstraction that is used only for resource management and control. Therein services may be found and interact with each other. From the viewpoint of the operating system, the VE is the principal responsible for the consumption and use of resources, the recipient of sanctions in the event of policy violations and the entity that is legal to receive authorisation when services accessing control interfaces. Similar conclusions may be found in [14] [15]. In other words, a VE provides a place where services may be instantiated and used by a community of users or groups of applications while staying isolated from others residing in different VEs. Within a VE many types of EEs with their instances may be combined to implement and/or instantiate a service.

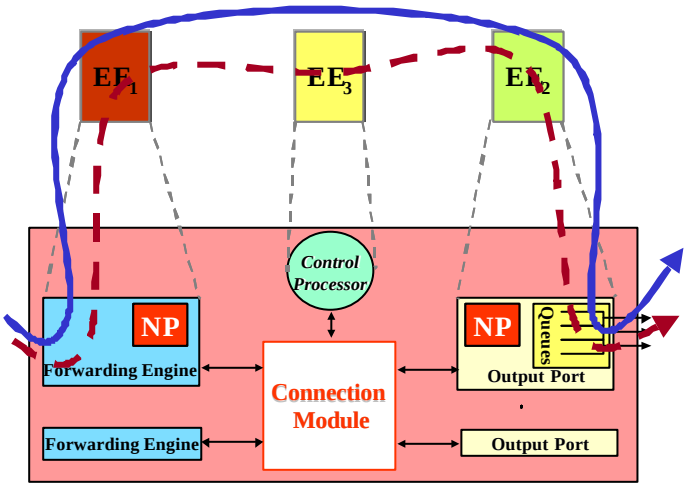


Figure 5-3: The Network Element representation.

Another property of the reference architecture is that it makes no assumptions about how "thin" a VE is. It may take the form of an application, or a specialised service environment e.g. video on demand, or even a fully-fledged network architecture as proposed in [16] [17]. Finally, a VE may coincide with an implementation (EE instance) that is based only on one technology e.g. Java technology. In either case this is a design decision dictated by customer requirements and/or the VE owner.

Out of all the VEs residing in a node there must be a privileged one that is instantiated automatically when the node is booted up and serves as a back door through which subsequent VEs may be created through the management plane. This privileged VE should be owned by the network provider, who has access rights to instantiate the requested VE on behalf of a customer through a VE Manager (VEM). From this viewpoint the creation of VEs becomes a kind of meta-service.

The other major and most important component of the reference architecture is the NodeOS. It offers all those facilities[4] that are necessary to keep all the other components together, and provides resource control, security, management, active service provisioning (ASP) of service components, and demultiplexing. More details may be found in [18] [19]. All these facilities in the NodeOS co-operate to deliver the overall functionality of the NodeOS to achieve its goals.

Between VEs and NodeOS lies the node interface that encapsulates all the capabilities offered by the NE. Its objective is to create programmable abstractions of the underlying NE resources, whereby third-party service providers, network administrators, network programmers or application developers can exert or extend node control through the use of higher-level API's. This interface coincides with the L-interface and its specification must be implemented by EEs in order to achieve interoperability among different NEs. Finally, between the NodeOS and the hardware NE there might be the open router interface. Its scope coincides with the scope of the CCM interface of P1520

The FAIN reference architecture is the starting point from which a detailed node architecture specification follows. Accordingly, it is complemented by the system architecture requirements, design and specification. This, together with customer/user/application requirements determines the degree of programmability to be built in the NE and the choice of technologies.

The previous two ingredients, namely the EE instances and the open interfaces require a NE to reside in. Packets arriving at the node have to follow different data-paths inside the node. At every part of the node, EEs have been instantiated implementing the programming methodology of their corresponding EE types with some of them creating component-based programming environments. This gives rise to a new generation of network elements with architectures that are component-based. Such trend has been accelerated by the advent of innovative network products like the Network Processors (NP), which are capable of hosting an EE without the cost of performance degradation. Figure 5-3 depicts this new situation in the form of a possible NE representation.

In FAIN we have designed and built a prototype of an AN node that adopts the scenario above. Instead of an NP we have built one EE at the kernel space and another one at the user space. Both EEs support the building block approach and receive packets, which are then directed to specific components as part of their data-path NE traversing. More detailed description may be found in [18].

5.1 Discussion on the FAIN Reference Architecture

The FAIN NE reference architecture serves as a way to manage and control overall service deployment. Based on the ability to combine different EEs as part of the service creation and deployment not only may specific service components be deployed but also the whole programming environment (EE instance) which is bound with existing EE instances. To this end, different functional models may be mapped onto the same physical NE infrastructure. One example could be that an EE instance is deployed in an NP while another is represented by an ASIC. This constitutes a departure from the Active Networks reference architecture where only EE instances of the same type are allowed to communicate.

Furthermore, the separation between VE and EE allowed us to separate the resource control from the specifics of a technology used by EEs as well as multiple EEs may be hosted by one VE and still being able to allocate resources as these are assigned to VE.

[4] We use here the word facilities to refer to services offered by the NodeOS to VEs and distinguish from services found inside EEs.

Returning back to the ForCES working group and in particular their architectural representation of an NE built around the CEs and FEs as well as the proposed FE model [13], it is clear that the EE definition in FAIN is also valid for an FE definition as inferred from the current state of the IETF working group. In addition, an EE that resides in the control plane may well represent CEs since such control EEs are used for controlling EEs in the transport plane.

Accordingly, the issues of FE control and configuration, especially those that pertain to dynamically extensible FEs, are identical with those in FAIN. As such the mechanisms for service deployment built in FAIN which facilitate configuration and control of EEs (in the transport plane) may also be used for the same purposes within the context of the ForCES activity.

In the sections that follow we are going to describe in details how the reference architecture was realised to create the FAIN network and node architecture.

6 FAIN NETWORKING ARCHITECTURE

6.1 NETWORKING ISSUES IN FAIN

The FAIN active network architecture defines active nodes, which provide full flexibility to the user for network management and service provisioning. The defining characteristic of an active node is the ability for users to load and manage software components dynamically and efficiently. This can be achieved safely since customers who are sharing the same active node would be provided with a VPN-like resource partitioning.

Packets requiring active processing are marked 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 (element and network management nodes). Access to this functionality will be controlled and only possible via a well-defined API.

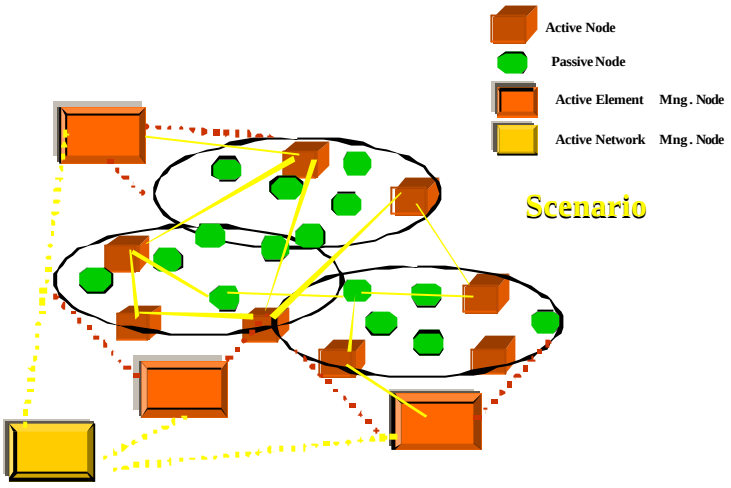


Figure 6-1: FAIN Active Networks

Figure 6-1 exemplifies a configuration of an active network and its management nodes.

6.2 COMPONENTS IN THE FAIN ACTIVE NETWORK

The FAIN Reference Architecture consists mainly of the following entities*:*

1. Node Systems
2. Management Systems

The FAIN Node systems consist of:

- *Active Applications/Services* (AA) are applications executed in active nodes.
- *Execution Environments* (EE) are environments where application code is executed. A privileged EE manages and controls the active node and it provides the environment where network policies are executed. Multiple and different types of EE are envisaged in FAIN. EEs are classified into virtual environments (VEs), where services can be found and interact with each other. VEs are interconnected to form a truly virtual network.
- *NodeOS* is an operating system for active nodes and includes facilities for setting up and management of communications channels for inter–EEs and AA/EEs, manages the router resources, provides APIs for AA/EEs, and isolates EEs from each other. Through its extensions the NodeOS offers facilities through the following components*:*

- *Resource Control Facilities* (RCF). Through resource control, resource partitioning is provided and VEs are guaranteed that consumption stays within the agreed contract during an admission control phase, whether static or dynamic.
- *Security Facilities* (SF). The main security aspects are authentication and authorisation to control the use of resources and other objects of the node such as interfaces and directories. Security is enforced according to the policy profile of each VE.
- *Application/Service code deployment facilities* (ASP support). As flexibility is one of the requirements for programmable networks, partly realised as static or dynamic service provisioning, the NodeOS must support code deployment.
- *Demultiplexing facilities* (DEMUX). As flows of packets arrive at the node, Demultiplexing filters, classifies and diverts active packets to the appropriate VE, and consequently to the destination service inside the VE.
- *Node Management Facilities* (NM). The main aspects are the initiation and maintenance of VEs, control and management of the RCF and SF, management of the mapping of external to node policies into node resource and security configurations.

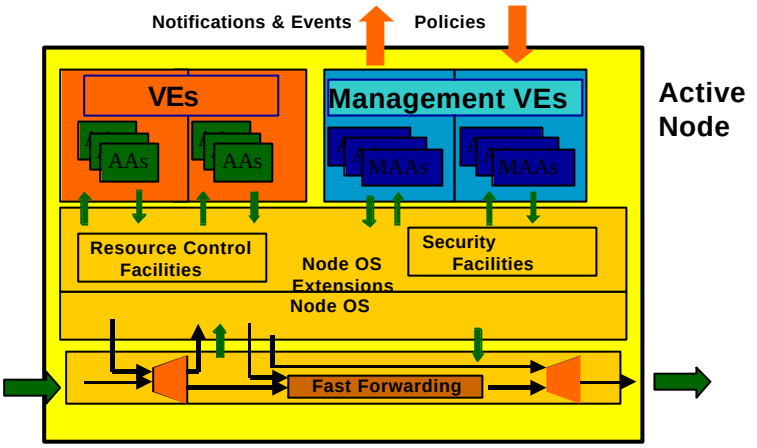


Figure 6-2: FAIN Active Node

Figure 6-2 describes the main design features and the components of the FAIN nodes:

In FAIN, node prototypes that are under development include: a high performance active node, with a target of 150 Mb/s; and a range of flexible and very functional active nodes/servers, with the target on multiple VEs hosting difference EEs

The common part of the prototypes (the FAIN middleware) is the NodeOS with the relevant extensions. Further details and discussions about the active node are provided in the remaining of this deliverable and in D2.

The management approach in the FAIN project, which takes policy-based approach.

The FAIN Management systems consist of

- *Policies:* Description of policies required to manage the active nodes and network
- *Node management component:* Design of management components within the active nodes, which will execute policies within an active node and monitor the local node resource usage. The execution of policies means mapping target policies into node resource configurations

- *Management stations:* 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 as well as managing the network service provisioning process.

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 see the need for two types of management nodes in order to provide these mechanisms:

- Element Management Stations (EMS)
- Network Management Stations (NMS)

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 in Figure 6-3.

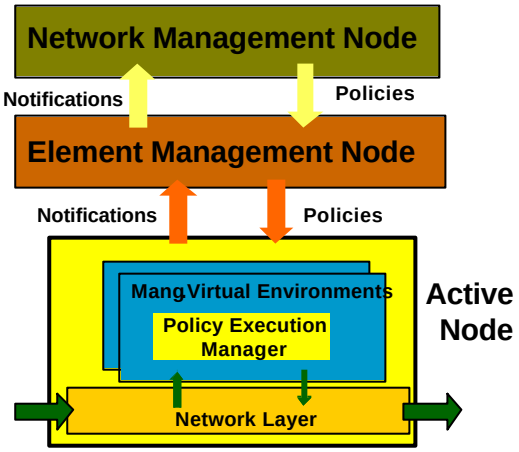
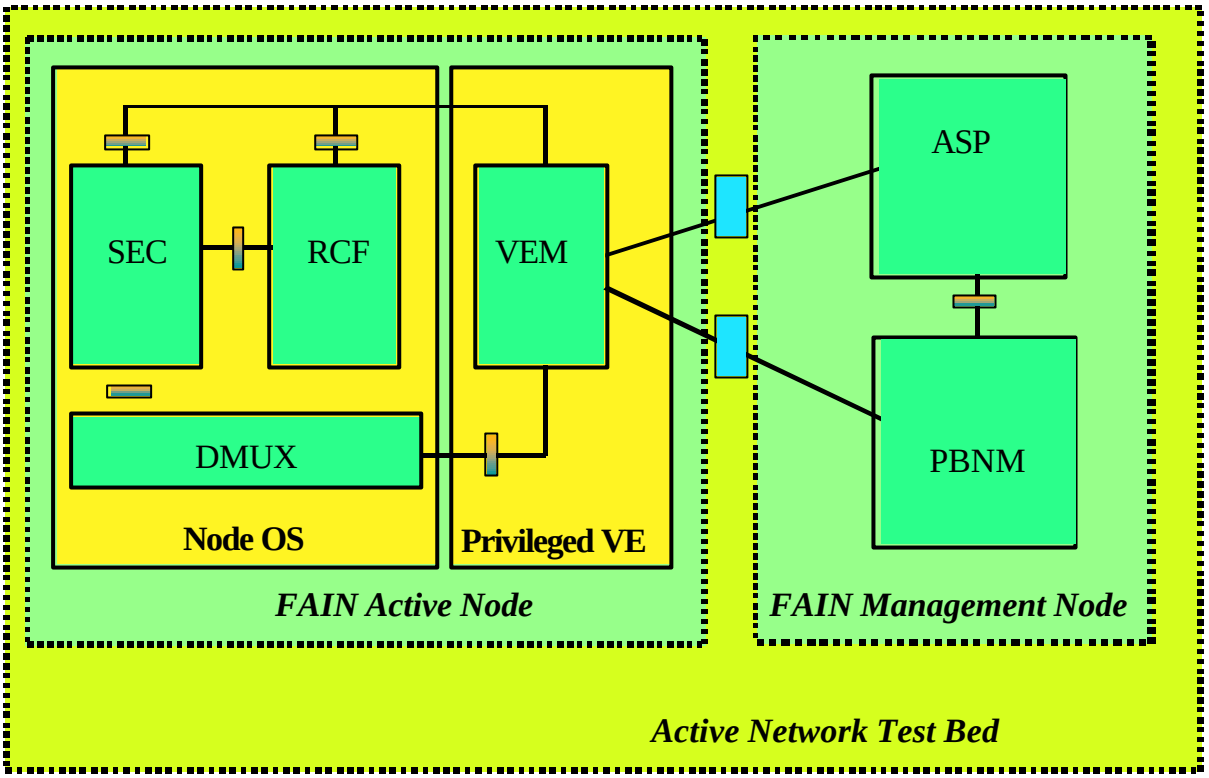


Figure 6-3: Active Network Management

Furthermore, the relationships between the EMS, NMS, and active nodes with regards to the policy flow are shown in Figure 6-3.



7 FAIN NODES

Figure 7-1: Overview of AN node architecture

Figure 7-1 provides an overview of the major AN node components and their corresponding interfaces that comprise the FAIN AN node architecture. More specifically, the Privileged Virtual Environment has been enhanced with a new component, called VE manager, which implements the VE management framework. This component is the most crucial one as it offers a number of node services that are deemed necessary to configure and setup the node. It is used for instantiating new VEs, deploying EEs and components therein as well as control interfaces that allows services inside VEs to customise resources according to application-specific requirements. In addition, the proposed framework allows the implementation of other components like resource managers in RCF or channel managers in DMUX to be easily integrated with the implementation of the framework by means of a set of classes from which these components inherit. Finally, the VEM manager specified another set of interfaces, namely the Template Manager and Component manager that facilitate communication and integration with types of EEs other than the one that the VEM used for its own implementation. This will enable future integration with other implementation instances that are currently under development.

The DMUX has been enhanced with a Channel manager that is capable of creating different types of Channels, which are used to receive different types of packets, e.g. data packets or ANEP packets, and consequently deliver these packets to the proper services that are running inside different VEs. These channels are created by the DMUX components and are given to the requested VEs, which control them. To this end, VEs may request the creation or deletion of channels as well as configure these channels to receive certain packets, e.g. a specific IP address. The DMUX actually provides the two-ends (input and output) of a plug and play datapath that is supported by the component-based AN node architecture.

The goals of security architectureas stated in D2 have stayed the same. The architecture itself was further developed and is now more precisely defined. Two new entities were added, namely, the Security Manager and the connection manager. The former impacted the authorisation functionality, which now supports multiple authorisation engines, and exports security interfaces (policy and credentials) to the node. The latter, provides security support for hop-by-hop data integrity over connections with adjacent nodes. To this end, the security related options of the ANEP header were also specified and used in scenarios that involve this aspect of security functionality.

The AN node resource control (RCF) has adopted the VEM framework whereby the resources and their corresponding resource managers of the original RCF architecture in D2 are encapsulated in the Components manager of the VEM. Accordingly, the RMs can be deployed and controlled as any other regular service component through the interfaces inherited from the Component Manager. The RCF has also been extended with an Admission Control entity that is responsible for deciding whether the new VE creation request may be accepted provided that there are resources available left in the AN node.

The AN node architecture has been implemented in a Java EE on top of a Linux operating system using the Netfilter for packet classification and forwarding. This implies that the management functionality of the node has also used a Java EE. However, another aspect of the architecture is the simultaneous support of multiple EEs provided by the VEM framework. To this end, we have built two different types of EEs; one high performance EE in the kernel space of the Linux operating system and one control EE, called Active SNMP, which is based on SNAP EE.

The high performance EE is capable of dynamically deploying service components in the datapath on behalf of different VEs. The Active SNMP EE is an example of in-band signalling EE that enables valid users to control node resources by communicating with an SNMP agent.

8 FAIN PBNM MANAGEMEN

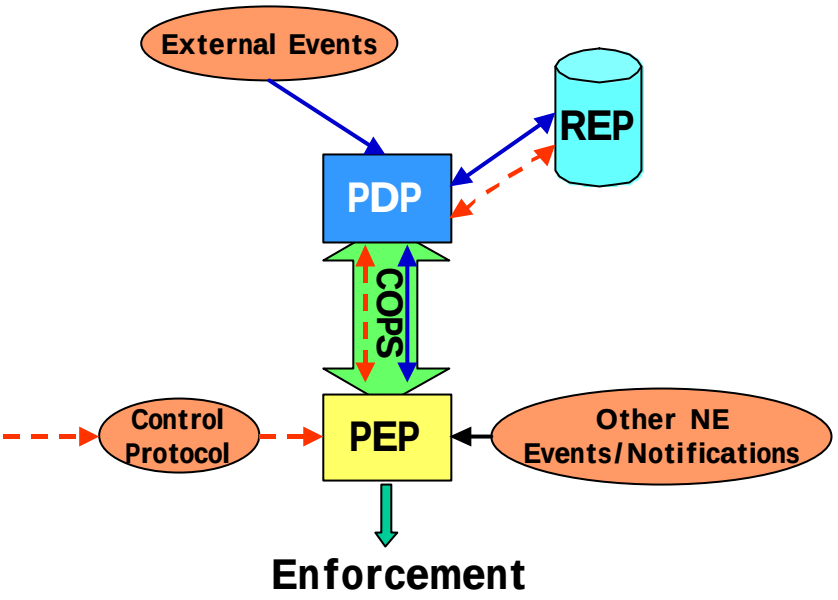


Figure 8-1 -The IETF Policy-Based Networking

Network management, either of telecommunication or data networks, has long been argued along the manager-agent model [1] and deals with three fundamental aspects: a) functionality grouped according to five areas, namely, Fault, Configuration, Accounting, Performance and Security (FCAPS), b) information modelling by which network and network element resources are identified and abstracted in a way that underpins specific operations semantics, and c) the communication method among managers and agents.

Realising and implementing this model gave rise to a series of different network management architectures each one being the result of limitations of the previous but most notably because of emerging network architectures and the new demands imposed on their management. The latter was the combined effect of rapidly changing customer requirements, enabling technologies, and new market forces. Such management architectures have been categorised into centralised, hierarchical and distributed while a hybrid between hierarchical and distributed is also possible [2], [3].

Although one of the original intentions of making the management architecture hierarchical was the need for reducing the management data flows from the agent to the central manager, the management application was still running in the manager and away from the Network Element (NE) while treating the agent as a mere implementation of the communication protocol unable to make any decisions. Management by Delegation (MbD), introduced first as a concept in [10], was conceived in an attempt to transfer the management logic from the central management system closer to the managed entity. This results in alleviating the central management system from the management burden.

Traditional network management techniques focused on the procedures that are required to carry out FCAPS functions. They have built the communication protocols and information models that are pervasive across a network so that the network can be managed as smoothly as possible. However, what they have not addressed was a framework for a high-level automated network management based on well-defined rules, which capture the semantics of the procedures that the network should adhere to. Policy-Based network management filled this gap.

The following requirements are considered [4] as being increasingly consistent across different market segments ranging from Small to Medium Enterprises to large Enterprises as well as Service Providers` Operation Support System (OSS) environments:

- Ease of use and implementation.
- Ease of integration.
- Dynamic adaptability
- Scalability
- Reliability
- Cost/Value

Policy-Based network management is also instrumental in meeting most of the aforementioned requirements as it provides a common "language" [5] to communicate these requirements to the network infrastructure, which is then configured accordingly. Policies capture the semantics of a second level and more advanced management logic, which can be associated with different uses of the network resources at the same time.

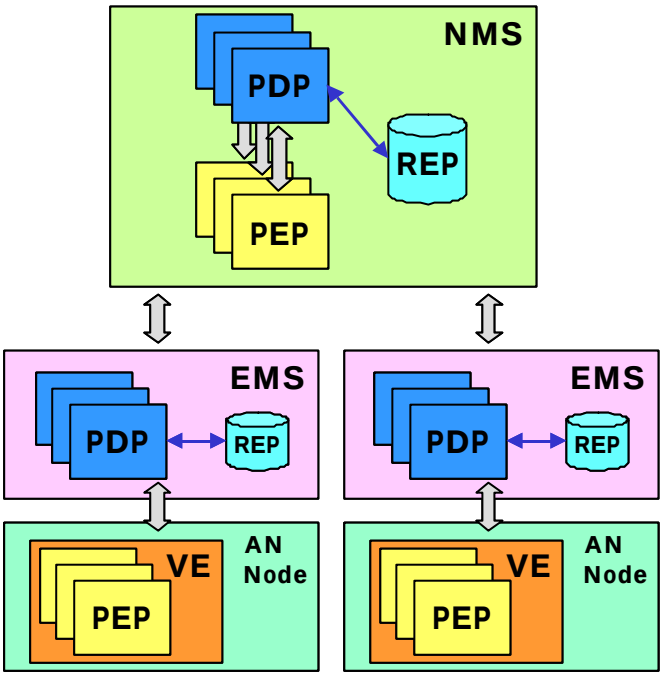


Figure 8-2: The hierarchical FAIN Management Architecture

Figure 1 abstracts the main entities that participate in Policy-Based networking. Central to this figure are the Policy Enforcement Point (PEP) that resides in the NE and the Policy Decision Point (PDP) that resides in a policy server [6]. A database, generally known as Policy Repository, stores the policies that the NEs must adhere to.

The PDPs and PEPs collaborate by exchanging communication messages by means of a standard protocol in order to support policy control. Such messages transport configuration information, syntactically and semantically defined in the PIB (Policy Information Base). Two common models are used for policy control: Outsourcing and Configuration [7].

In either of the two models, a large number of management operations may be automated and simplified making network management simpler and more scalable. In addition, policy aware NEs manifest a vendor's independence making integration and interoperation feasible without prohibiting product differentiation.

Finally, treating PEPs and PDPs as execution environments that can be extensible, like the elastic server in the MbD paradigm, functionality can be added dynamically thereby adapting the network to new demands in the form of new policies. This was one of the motivations to design our management based on the policy framework.

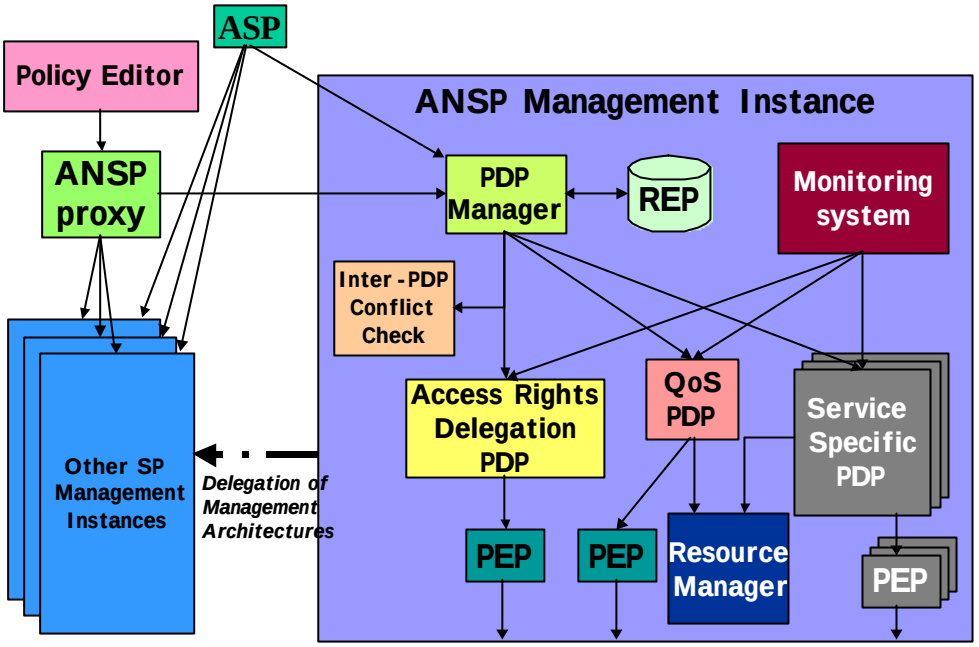


Figure 8-3: FAIN Management Instances and their Components

The FAIN PBNM management architecture is designed as a hierarchically distributed architecture, consisting of two levels (two-tiered architecture): the network management level, which encompasses the Network Management System (NMS) and the element management level, which encompasses the Element Management System (EMS).

Furthermore, the defined policies have been categorised according to the semantics of management operations, which may range from QoS operations to service-specific operations. Accordingly, policies that belong to a specific category are processed by dedicated Policy Decision Points (PDPs) and Policy Enforcement Points (PEPs) (Figure 8-2).

The NMS is the entry point of the management architecture. It is the recipient of policies, which may have been the result of network operator management decisions or of service level agreements (SLA) [8] between ANSP & SP, or SP & C. These SLAs require reconfiguration of the network, which is automated by means of policies sent to the NMS.

Network-level policies are processed by the NMS PDPs, which decide when policies can be enforced. When enforced, they are delivered to the NMS PEPs that map them to element level policies, which are, in turn, sent to the EMSs. EMS PDPs perform similar processes at the element level. Finally, the AN node PEPs execute the enforcement actions at the NE.

The use of this "policy control configuration model" [7] and its use in a hierarchically distributed management architecture combines the benefits of management automation with reduction of management traffic and distribution of tasks.

As the FAIN management architecture is based on the FAIN Business Model, the relationship among the three main actors, namely, ANSP, SP, and C, is projected directly onto the architecture. Accordingly, each one of these actors may request and get his own (virtual) management architecture through which he is enabled to manage the resources allocated to the Virtual Environments (VE) of his Virtual Network (Figure 8-2).

In this way, each actor is free to select and deploy his own model of managing the resources, namely his own management architecture, which can be centralised, hierarchical, policy or non-policy based. The complexity of the virtual network and the types of service that are deployed in it, dictate the particular choice of management architecture by its owner. In addition, different management architectures simultaneously coexist in the same physical network infrastructure as they may be deployed by different actors. To this end, we create an environment that is capable of accommodating

opposing requirements, an accomplishment that is beyond the capabilities of the traditional approach of monolithic architectures.

Our model extends the Tempest approach [9] to the management plane, which was the first to advocate the simultaneous support of (virtual) control architectures for ATM networks.

It also extends the scope of Management by Delegation (MbD) [10] as it allows delegation of the network management responsibility to a third party, e.g. an SP, which can be deployed and hosted in a separate physical location from the NMS of the owner of the network, e.g. the ANSP.

Figure 8-3 illustrates the aforementioned discussion. Starting with the management architecture of the network operator, namely the ANSP, it instantiates and registers a new Management Instance (MI), which is delegated to one of his customers, i.e. the SP. This management instance will host the SP's management architecture. The SP has the option to buy from the ANSP an instance of the ANSP's architecture, in our case a policy-based one. To this end, the network management architecture developed by the ANSP is not only used for managing the Network Elements (NEs) but it becomes a commodity, thus creating another important source of income for the ANSP.

Furthermore, the ability of the ANSP to generate and support multiple management domains may create additional business opportunities. For example, the ANSP may build an OSS hosting facility for SPs to instantiate their own management architectures. In this way, the ANSP may sell both his expertise in running and operating an OSS as well as the architecture and its corresponding implementation.

In contrast, the SP does not need to build his management architecture from scratch but can customise an existing one according to the services he intends to run. Alternatively, the SP may deploy his own management architecture using the OSS hosting facility provided by the ANSP, so reducing the cost of managing the network.

In FAIN we have focused and experimented with the automated instantiation of management architectures using as a blueprint the PBNM system of the ANSP to instantiate another management system for the SP. Note also that this instantiation relationship can be recursive in the sense that the SP may further delegate his own instances to a Consumer.

Finally, the architecture of the MI used by the ANSP has been designed in such a way that it is dynamically extensible in terms of its functionality, as a result of using active networks technology.

The ANSP's management architecture can be extended in two distinct ways:

a) Deployment of a whole new pair of PDP/PEPs that implement new management functionality.

b) Extension of the inherent functionality of existing PDP/PEPs.

The former is triggered by the PDP Manager whereas the latter is achieved by the PDPs themselves. The execution of the extension, namely fetching and deploying the requested functionality, is the responsibility of FAIN's Active Service Provisioning (ASP) system [D8].

One important assumption underlying the previously described virtual management architectures is that well-established open interfaces and protocols have to be provided by the NEs. This may seem from the outset to be a demanding condition but there is convincing evidence that there exists a strong push towards ubiquitous open interfaces. Initiatives like the IEEE P1520 and lately the IETF ForCES working group serve as a proof for such claims. Furthermore, the programmable and active networks paradigm also relies on similar assumptions [D7-]. These requirements are for non-FAIN active nodes, FAIN AN follows this assumption.

9 FAIN ACTIVE SERVICE PROVISIONING

Active Service Provisioning, or ASP for short, is understood in the context of the FAIN project as system aiming at deploying active services in the FAIN network. In general, active service deployment is considered as a process of making a service available in the active network so that the service user can use it. The deployment process is usually seen as a number of preparatory activities before the phase of the service operation. The typical activities include releasing the service code, distributing the service code to the target location, installing it and activating it.

Since the mid nineteen-nineties many efforts have been made to develop Active Networks technology to enable more flexibility in provisioning services in networks. By defining an open environment on network nodes this technology allows to rapidly deploy new services which otherwise may need a long time and adoption of hardware.

The FAIN project follows an approach in which a number of existing and emerging active network technologies are integrated. With regard to deployment, it proposes a novel approach to deploying services in heterogeneous active networks. In particular, the FAIN approach to deployment is characterised by the following:

- On-demand service deployment support. The ASP supports deploying a service whenever it is needed. A service deployment may be explicitly requested by a service provider or by another service already deployed or a management component.
- Component-based approach. Deploying and managing high-level services requires an appropriate service model. While fully-fledged component-based service models are an integral part of many enterprise computing architectures (e.g. Enterprise JAVA Beans, CORBA Component Model, Microsoft's.NET), it is not the case in many approaches developed by the active networking community. The FAIN deployment framework is designed on top of a component-based service model similar to the CORBA Component Model. The service model is hierarchical in that service components may recursively include sub-components. This allows for a fine-grained service description and composition.
- Network and node level architecture. To deal with complexity of deployment issues in active networks, the Active Service Provisioning has been designed according with rule of separation of concerns. The network-level ASP copes with network issues that include finding the nodes of the target environment for a given service considering topological service requirements as well as network link QoS requirements, for instance bandwidth. The node-level ASP, on the other hand, is concerned with node specific requirements, including technology and other service dependencies.
- Integrated Service Deployment and Management. The FAIN approach to service deployment is tightly integrated to FAIN service and network management. On one hand, the ASP depends on the service management framework implementing EE-specific deployment mechanisms, including installation and instantiation. On the other hand, the target environment in which the service is to be deployed are co-determined by the Network Management System The target environment is defined to be a Virtual Active Network which is established by the FAIN Network Management System. The VAN is created by the management system according to the service requirements.
- Selective code deployment. The service code distribution is done by selective downloading selected code modules from a code repository. The decision as to which code modules are needed is made at the ASP components at the target active nodes.
- Support for heterogeneous services and networks. The ASP has been designed to enable service deployment in heterogeneous networks. This is achieved by specifying an unified interface to the node capabilities and a unified notation for describing service specification and the implementation requirements. Whereas the CORBA technology is used to define the unified API to the node, the XML technology is used to define the unified service description.

The main actors communicating with the ASP system are:

- **Service Provider,** or SP for short, composes services that include active components and deploys these components in the network via the Active Service Provisioning, and offers the resulting service to Consumers. The service provider is responsible for releasing and withdrawing a service which includes a service version update or a complete remove of the service from specific nodes or from the complete active network respectively. Furthermore, the SP may be represented by the FAIN Network Management System with regard to initiation of service deployment or service reconfiguration.
- **Active Network Service Provider,** or ANSP for short, provides facilities for the deployment and operation of the active components into the network. Such facilities come in the form of an active middleware, support of new technologies. ANSP is represented by Active Nodes which are the target environment in context of deployment, which means that services may be deployed in these nodes and use the node resources made available to them by the ANSP.

These roles are described in the FAIN Enterprise Model in [11] in more detail. The main use cases of the ASP system are:

- Releasing a service. The Service Provider who decides to offer his service in the active network has to release it in the active network. The service is released by making the service meta-information and service code modules available to the ASP system.
- Deploying a service. After the service is released in the network, the Service Provider may want to deploy his service so that it can be used by a given service user. It means finding a target nodes that are most suitable for the given service installation, determining a mapping of the service components to the available Execution Environments of the target node, downloading the appropriate code modules, and finally installing and activating them.
- Reconfigure Service. The Service Provider or Network Management System on his behalf may request changing the current configuration of the service. It may include modifying component bindings, deploying additional service components or redeploying components that have been already deployed.
- Removing a service. The Service Provider may request to remove a deployed service from the environment it was deployed in. The ASP identifies the installed service components and removes them from the EEs of the target environment.
- Withdrawing a service. A service released in the active network may be withdrawn so that is no longer available to be deployed. The ASP removes the service meta-information and discards the service code modules.

The FAIN ASP system has a two-layered architecture: the network level and node level. The network level functionality is concerned with finding the target nodes for the service to deploy, coordination of the deployment process at the node level, and providing a service code retrieval infrastructure. At the node level, necessary service components are identified through code dependency resolution as well as the deployment mechanisms, including service installation, activation and pre-configuration are controlled.

Network Level ASP Design. The network level ASP system consists of three components depicted in Figure 9-1: Network ASP manager, Service Registry and Service Repository.

The Network ASP Manager serves as an access component to the ASP system. In order to initiate the deployment of an particular service, a Service Provider contacts the Network ASP Manager and requests a service to be deployed as specified by the service descriptor.

The Service Registry is used to manage service descriptors. Service descriptors are stored on it, when a service component is released in the network. Network ASP Manager and the Service Creation Engine (described below) may contact the Service Registry to fetch service descriptors. Finally, service descriptors are deleted from the Service Registry, if a service is withdrawn from the network. In figure 3 only one Service Registry is shown in the network. Of course, several instances with possibly different content could be deployed in a network.

The Service Repository is a server for code modules. A code module is stored on the Service Repository, when a service descriptor referencing the particular code module is released in the network. The Code Manager, which is part of the node level ASP system and is described below, may fetch code modules from the Service Repository. A code module is deleted, if a service descriptor referencing the particular code module is withdrawn. As is the case for the Service Registry, several Service Repositories may coexist in a big network.

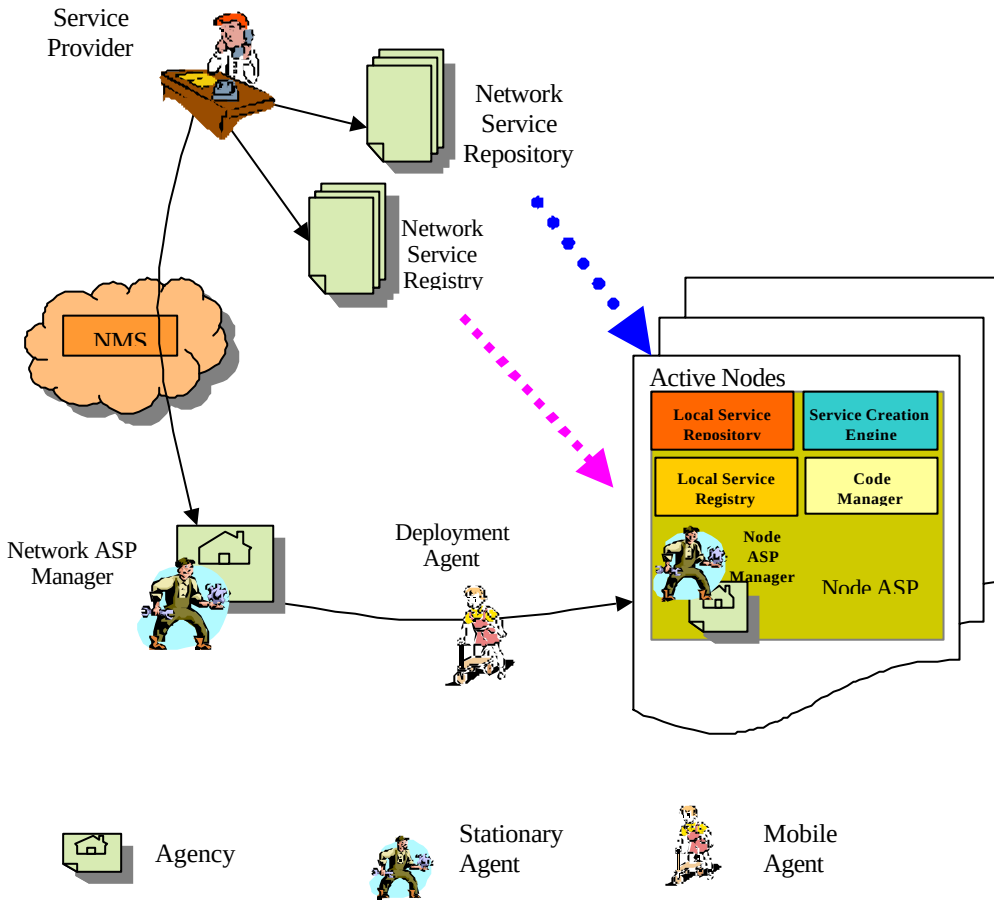


Figure 9-1: The FAIN Active Service Provisioning

Node Level ASP Design. On the node level, the following components make up the ASP system as shown in the node ASP block in Figure 9-1: Node ASP manager, Service creation engine and Code Manager.

The Node ASP Manager is the peer component to the network ASP manager on the node level. The network ASP manager communicates with the node ASP manager in order to request the deployment, upgrading and removal of service components. The requests are dispatched to the service creation engine, or the code manager, respectively, which implement corresponding methods.

The Service Creation Engine plays a major role in the node level deployment of service components. Its main task is to select appropriate code modules to be installed on the node in order to perform the requested service functionality. The service creation engine matches service component requirements against node capabilities and performs the necessary dependency resolution. Since the service creation engine is implemented on each active node, active node manufacturers are enabled to optimise the mapping process for their particular node. In this way it is possible to exploit proprietary, advanced features of an active node. The selection of service components is based on service descriptors that are retrieved from the service registry. As a result information about code modules that are to be installed on the particular node (a so-called service tree) is passed to the Code Manager.

The Code Manager performs the execution environment independent part of service component management. During the deployment phase, it fetches code modules identified by the service tree from the service repository. It also communicates with Node Management to perform EE-specific part of installation and instantiation of code modules. The Code Manager maintains a database containing information about installed code modules and their association with service components. If a particular service component needs to be removed this database is consulted in order to find out which code modules are associated with the component and, as a consequence, must be removed as well.

Please note that information fetched from service registry and repository is locally stored in their respective cache (local service registry, local service repository) in order to optimise recurrent service deployment requests.

10 FAIN TESTBED

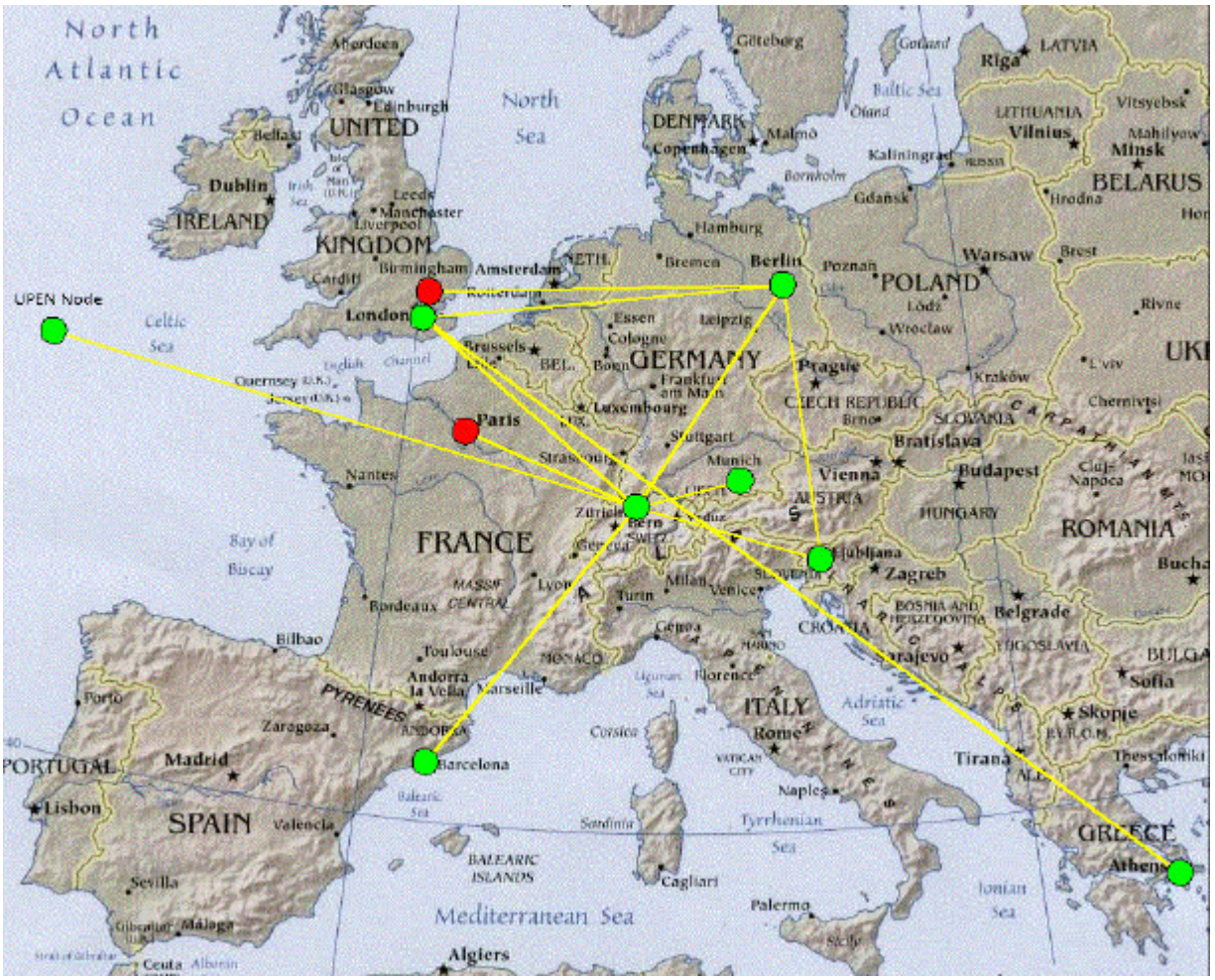


Figure 10-1: Testbed topology

This section provides an overview of the FAIN active testbed, which serves as a permanent experimental network for active network technologies up to the end of the FAIN project and possibly beyond. The testbed is completely operational. In the remainder of this section we will describe the structure of the testbed, will precise where the different facilities and components are located and briefly describe the type of nodes running at various sites.

10.1 NETWORK TOPOLOGY AND INTERCONNECTION

10.1.1 Testbed topology

Figure 10-1 depicts the current topology of FAIN testbed, with four sites (ETH, FHG, UCL, JSIS) forming two core triangles and the rest of the sites connected as leaves to one of the core nodes. The decision about which node had to be a core node and to which core node the other nodes had to refer to was taken after measurements of the bandwidth and link quality between the different sites. Essentially, this is a three-level hierarchical tree topology with cross connections at the second level of the tree. The advantage of this topology in comparison with the full mesh is that the later provides only single hop paths between active nodes, while it may be more interesting to test applications over multi-hop paths. On the other hand, a tree with cross connections provides alternate paths between nodes, which is not the case with a simple tree topology. Finally, contrary to full or partial mesh, a carefully constructed tree topology accommodates for the fact that some partners have a lower bandwidth connection to the testbed either due to technical limitations or due to corporate security policy. The complete network is shown in Figure 10-1

10.1.2 Tunnel configuration

The FAIN testbed has been set up as an overlay (i.e. virtual) network on the existing network infrastructure. The overlay network is based on IP tunnelling and is realised by appropriately configuring point-to-point tunnels between specific nodes. There are several different tunnelling technologies and the choice of tunnelling technology depends on the requirements. In FAIN, we have employed simple IP GRE tunnelling, since the major requirement is to prevent interference of experimental traffic from the production traffic. We do not consider testbed traffic to be of sensitive nature (confidential), so there is no need to use IPSEC tunnelling to protect this traffic while in transit over public Internet.

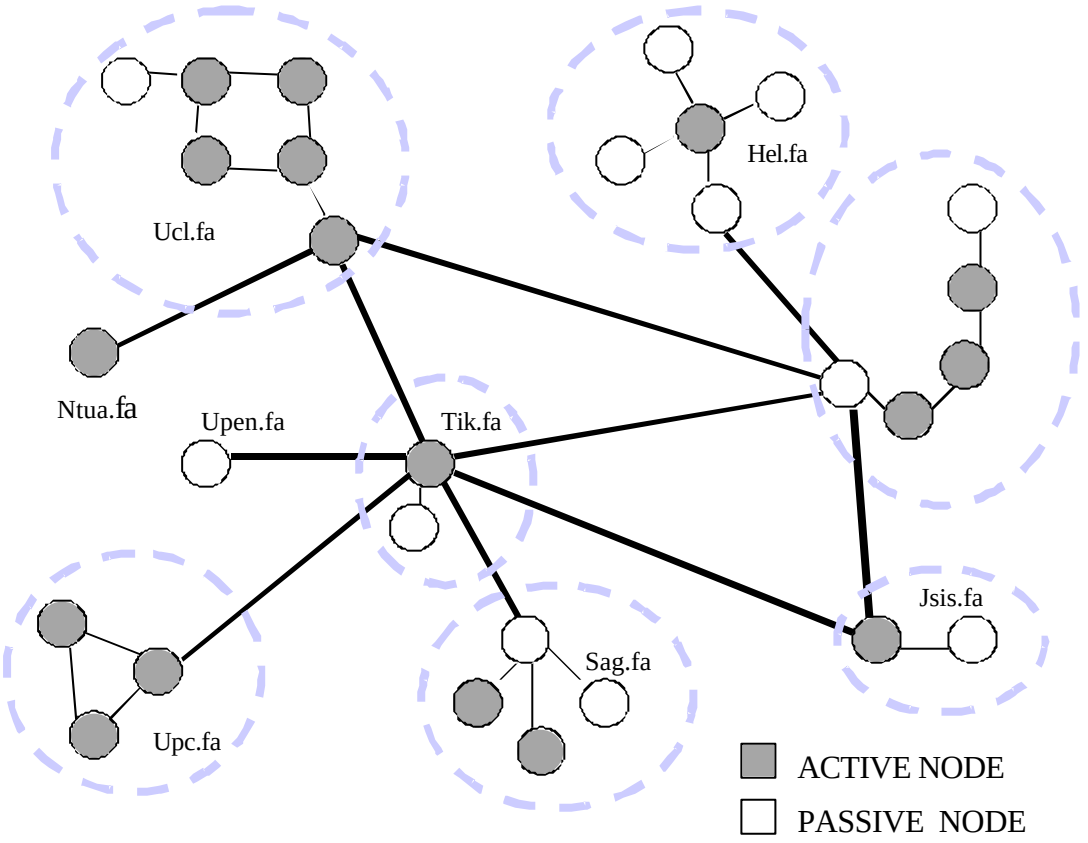


Figure 10-2: FAIN Nodes' overview

For the two tunnel endpoints, a properly configured tunnel looks the same as a physical point-to-point link, i.e. the nodes "think" they are directly connected, even though they use public Internet to communicate with each other.

10.1.3 Partner Network Data /Properties

Each partner site has:

- At least one node connected to the public internet acting as a tunnel endpoint
- A testbed subnet behind the tunnel endpoint with the address range of the form 10.0.p.0/24, where p is the partner number from Consortium partner list (in order of appearance in the FAIN Consortium partner list).

Partners can freely use the addresses from the private address range assigned to them.

10.1.4 Domain Name service

There is a DNS service running within the testbed. The primary DNS server is hosted and maintained by FHG in Berlin and its IP address is 10.0.12.12. A secondary DNS server is hosted at ETH in Zurich and has the IP address 10.0.11.11.

All nodes and hosts within the testbed use.fa as the top-level domain. The FQDN names for hosts within the testbed have the following form

hostname.FAIN-partner-code.fa

For example, the host "onizuka" located at FHG FOKUS is called onizuka.fhg.fa.

10.2 SITES OVERVIEW

Figure 10-2 shows the actual status of the FAIN testbed. The bold lines represent the tunnels whereas the lighter lines are the links in the private networks at the partners' sites. The FAIN testbed comprises different types of FAIN nodes:

- FAIN Active Network Nodes, either type A (VEM+Demux,+RCF_+Sec +ASNMP +PromethOS-based) or type C (hybrid router, which combine a commercial router with an active network EE provided by a physically separate and attached PC)
- FAIN Element Management Station (EMS)
- FAIN Network Management Station (NMS)

EMS and NMS could be installed on either active or passive nodes.

10.3 MONITORING TOOL

The FAIN testbed is constantly monitored using a "ping-based" tool. It checks whether the tunnel endpoints and the most important active nodes are up, the ports where the most important service run are open, and monitors the average delay, loss rate and bandwidth on the links.

11 FAIN SCENARIOS

The purpose of the application scenarios is to outline the functional concepts of FAIN in an intuitive and realistic manner. The FAIN application scenarios are therefore placed in circumstances that reflect requirements and demands of reality.

The application scenarios defined in FAIN are:

- DiffServ Scenario
- WebTV Scenario
- Web Service Distribution Scenario
- Video on Demand Scenario
- Mobile FAIN Demonstrator Scenario
- Managed Access Scenario
- Security Scenario

11.1 DIFFSERV SCENARIO

In this application scenario, a service provider (SP) agrees on a contract with an ANSP. The contract ensures him three levels of priority transmissions, each identified by a distinct DiffServ Code Point (DSCP).
The SP interconnects the three parties A, B, C as shown in Figure 11-1. Traffic entering or leaving A and C pass through the hybrid routers HANN-1 and HANN-2 respectively.

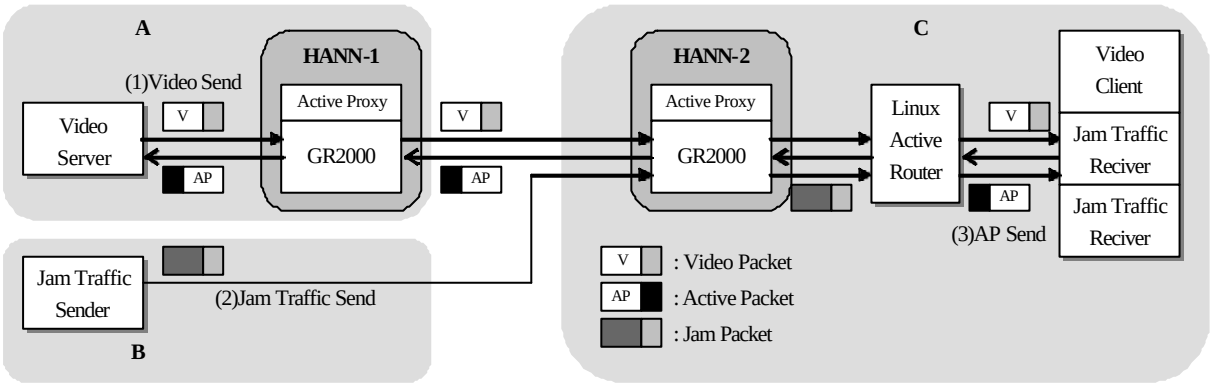


Figure 11-1: DiffServ Demonstration Scenario.

Party A sends a video to party C with low transmission priority. Some time later party B starts sending 'jamming' traffic to party C. The jamming traffic is identified by a DSCP guaranteeing higher priority over the video traffic. As long as the jamming traffic doesn't exceed the output bandwidth of HANN-2 nothing happens. As soon as it exceeds the output bandwidth, user C sends an active packet to A. The active packet is authenticated and authorized at each active node, which it traverses (i.e. at HANN-2 and HANN-1). Upon arrival at A a SNAP program changes the DSCP of the video stream, so that it gets higher priority over the jamming traffic.

11.2 WEBTV SCENARIO

An SP, called WebTV-SP, offers a WebTV service to its customers by broadcasting the video program over the Internet. The customers are able to watch the video irrespective of their terminal capabilities. In order to do so, the WebTV-SP requests from the ANSP to set up an Active Virtual Private Network (AVPN) wherein he can deploy services that are customised to customer requirements. Customers need only to subscribe to the WebTV service by contacting a server of the WebTV-SP.

In the scenario, one customer uses a terminal that is not capable of displaying the video stream correctly, e.g. it uses a handheld device with low processing power and low access bandwidth. The WebTV-SP pre-processes the video stream for this particular customer by transcoding the stream into a format understood by the handheld.

Based on a SLA between the ANSP and the SP, policies are sent to the ANSP MI. As a result the ANSP PBNM receives a QoS policy and enforces it on both the NMS and at all appropriated EMS. The active node management framework creates a new Virtual Environment (VE) for the WebTV-SP. If the VE creation is successful the ANSP PBNM enforces a delegation policy through the NMS and all appropriated EMS. The enforcement requests the active node management system to activate the newly created VE.

The ANSP creates a Management Instance (MI) in all the appropriate EMS stations for this WebTV-SP and sets access rights at the active nodes interfaces. The WebTV-SP is now ready to configure his AVPN by sending policies that are specific to customers. The WebTV- SP also installs the transcoder and duplicator service components. In addition, the WebTV-SP deploys service-specific policies in the WebTV-SP PDP of its MI, so that he can define its own service-specific policies that will be enforced in the active node. As the last step the monitoring system is used for the reconfiguration of the transcoder at runtime. This if used when the access bandwidth changes dramatically and the end-user needs a different transcoding format on the video stream for example.

The transcoder and the controller components are deployed via the ASP mechanism upon request in the SP's VE. The deployment is triggered by the SIP proxy when the SIP request from the customer arrives.

The creation of the VAN for the WebTV-SP entails that a management instance is created and configured. There is one MI at network level and one at element level. At the beginning only the PDP Manager is instantiated inside each MI. When requests are forwarded to the MI, the PDP Manager may decide to trigger the deployment of specialised functional domains by contacting the ASP in order to deploy SP PDP into the EMS, and SP PEP into the SP-VE.

At bootstrap of this SP PDP a set of alarms/event is configured by means of policies, which configure the monitoring system in order to receive those events/alarms from transcoder controller.

11.3 WEB SERVICE DISTRIBUTION SCENARIO

In the Web Service Distribution Scenario, Web (i.e. HTTP) traffic is distributed and redirected within the network among several distributed servers in order to provide reliability, performance and scalability for web services.

The web service infrastructure exploits the capabilities of AN technology as follows:

A Web service provider has access to Active Nodes, called "service nodes" or "Active Web Servers". The provider can install content and service logic onto those nodes and by such means he can implement features such as content distribution and personalisation, aggregation of user replies or fast response times.

Other active nodes, called "redirection nodes" are also available. They provide features, which allow filtering HTTP traffic, to build service sessions (i.e. to deal with per-user state) and to

forward the traffic of a particular session to a service node. Onto this nodes, code is downloaded which observes the network load, observes the load and availability of servers and based on this information determines a strategy for redirecting the traffic to the service node which is most suitable for a given web service/user.

The overall scenario is depicted in Figure 11-2. Both redirection nodes and service nodes will usually be physically located within the access network of the end user, the access network of the web service provider, or connected via a separated access network. For this reason, we may also call both of them "Active Web Gateways". Note that we assume that the core network is non-active and only provides basic IP connectivity.

The overall setting is fully transparent to the end user, i.e., the end user uses the web service via an ordinary web browser, using standard protocols such as IP or HTTP.

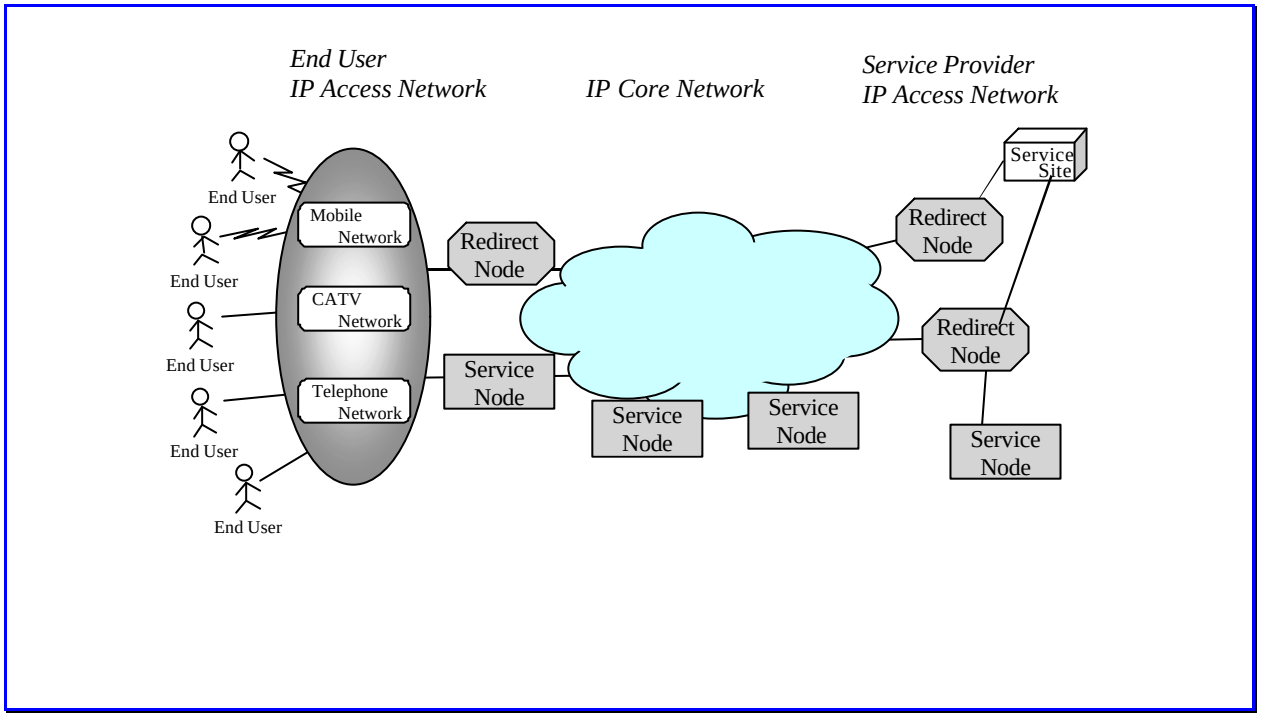


Figure 11-2: Service nodes and redirect servers implement active web services

The benefits of AN technology shown by the scenario include among others:

Network aware web services: Contrary to existing web services, they can operate based on information such as available bandwidth on access and network links, network topology, and load of servers. Because the information is not available at the client- or server side, it should be implemented with AN technology within the network.

Ease of service programming and management: Active networks eliminate the need for cumbersome ad-hoc solutions to specific problems, which require separate management; active networks provide common ground for deployment of new services and mechanism inside the network and unify the management of these mechanism and services.

Distribution of service logic: Service logic is executed at several locations, including specific points inside the active network, which is potentially advantageous for large volume services, fine (per user) granularity of services, new service features, etc. With existing solutions such as caches or content distribution networks, only content, but not service logic can be located within the network.

Dynamic, autonomous adaptation: The task of operators on service site (service provider, network provider) is getting bigger as the number of network customers/consumers is growing. So the task of provisioning should be dynamic. Besides active nodes may cooperate with each other.

11.4 VIDEO ON DEMAND SCENARIO

PromethOS is a Linux kernel-space NodeOS for active nodes. It is managed from Execution Environments (EE) in user space. Since the Virtual Environment Manager (VEM), which is an EE in user space, is also concerned with node management issues, an integration of both, VEM and PromethOS, becomes indispensable for the interoperability of different EE types. The video on demand scenario demonstrates this integration of VEM and PromethOS. It shows how the VEM management interface of PromethOS' user space library has been enhanced in order to control PromethOS.

The ANNs have been designed to support multiple VEs, EEs and EE instances. The scenario demonstrates therefore also how PromethOS is supporting multiple VEs and how it is able to differentiate among the customers by using those VEs. For the scenario, one EE per VE is instantiated in kernel space. The EE runs a Wave Video plug-in, which scales its functionality according to the different requirements of the customer.

At one site (e.g. service provider) an active node and a source for a video stream are located. At a different location, the video receiver is installed. Between the source and the active node, a high-bandwidth link provides sufficient bandwidth. The link models a high-bandwidth backbone. The active network node and the video receiver are connected by two links with different bandwidth. The capacities of the links reflect different Service Level Agreements. The active network node is supposed to adapt the high bandwidth requiring input stream according to the pre-set output capacities of the output streams.

At boot time, the ANN is running the VEM and the management components of PromethOS. As a customer request arrives at the ANN, the VEM orders the deployment of a VE, i.e. it assigns resources to the customer, and it orders the deployment of an EE where the Wave Video scaling plug-in is installed.

The request to initiate the service deployment is implemented by the FAIN-WP4 service specification. The service specification is parsed and resolved by the Service Creation Engine. The code required for the Wave Video plug-in is fetched from the code server and deployed on the ANN.

As a second request form a different customer arrives, the ANN creates a second instance with the same configuration but with different resources assigned to the VE. No additional code fetching is required anymore, since the code is available from the nodes local cache.

The creation process is fully implemented and controlled by the VEM. As the process completes, the video source gets a signal to begin with the transmission of the video flows.

11.5 MOBILE FAIN DEMONSTRATOR

This generic application scenario introduces a "Wireless LAN" showcase, which is implemented within FAIN. It shows how mobile networks benefit from FAIN concepts. It describes the idea of a "FAIN Dino Park".

The showcase "FAIN Dino Park" was made up, because it shows an interesting and promising use of mobile wireless network technology within the edutainment domain. Especially for mobile wireless networks where the bandwidth isn't abundant, the FAIN concepts exploit their advantages. The use cases of the "FAIN Dino Park" demonstrate how in a challenging wireless environment load balancing and load reduction approaches succeed in avoiding bottlenecks and how they could improve edutainment concepts.

The focus of the demonstration is on load balancing and load distribution in mobile networks. The show case is motivated by already installed famous amusement parks, but additionally it is equipped with a WLAN infrastructure based on products commonplace in the market. To realize load balancing and load reduction concepts, some additional software components are implemented and installed on the servers.

In total, there are 3 different use cases, which demonstrate load balancing and load reduction. Each use case is illustrated by demos. The use cases and the related demos are listed in the following.

- Use Case 1: shows a redirection of a connection. The connection to a heavily loaded access point is terminated and a new one is built up between the connecting client and a less burdened access point during peak-period demand whiles the registration.
- Use Case 2: This is a similar use case. It shows a simple redirection of a connection request. The connection is built up between the requesting client and a less burdened access point. The load at the burdened access point results by multiple request of video streaming.
- Use Case 3: This generic application scenario shows how a personal mobile SW proxy contributes to load balancing by mechanism of load reduction in the overall show case of the FAIN Dino Park.

Each of the cases relies on following components:

- A WLAN Access Controller, which is the central Control Unit. It is a software component and is deployed on a FAIN PromethOS active node.
- At least two WLAN access points, to which the clients are linked by a wireless link. Via these the clients receive data from a content server.
- At least one FAIN active node, which is PromethOS capable. Here, the following components are deployed:
- PromethOS: A framework for the manipulation of Linux Kernel modules (PromethOS modules) and for redirecting network traffic to such a PromethOS module.
    - Netfilter: A framework for the manipulation of network packets. Netfilter is used by PromethOS.
    - Iptables: This is the user space part of Netfilter. It is used to load and configure Netfilter modules and to redirect network traffic to such a module. PromethOS uses an extended version of Iptables, which is able to load and configure PromethOS modules.
    - PromethOS modules used by WLAN access controller: A PromethOS module is used for the actual manipulation and the monitoring of the network traffic.
- User interface: It is used to configure and monitor the PromethOS module.
- A server providing content used for the demonstration of actual data transfer via the WLAN access points. In the context of FAIN Dino Park, such a content server is dedicated to an exponent or specifically a designated server for registration procedures at the entrance.
- At least two terminals, which are notebooks or may be PDAs, each with WLAN capabilities. The terminals are equipped with a FAIN Terminal Daemon.

Optionally a load generator is required creating the background traffic which is used to trigger decisions on load balancing. Instead of using a load generator, the handover between access points may be demonstrated by simulating the load in the WLAN access controller.

The general infrastructure of the demos is shown in Figure 11-3:

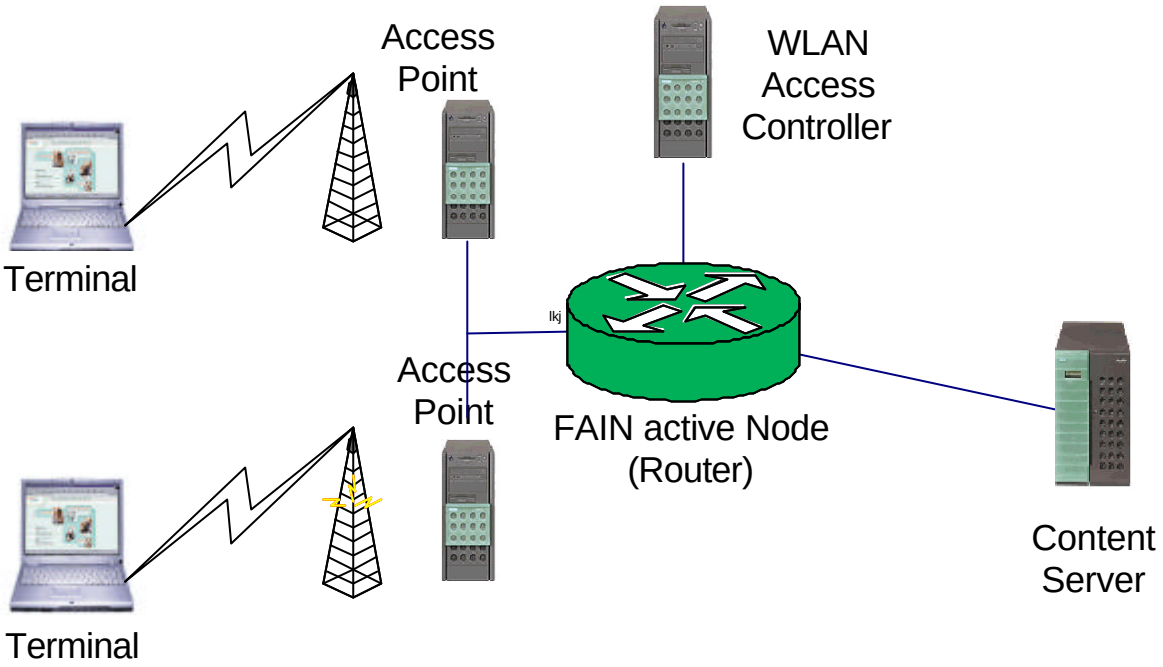


Figure 11-3: General Infrastructure of the Demos

11.6 MANAGED ACCESS

A host joining a network is assigned a set of network services according to some SLA, e.g. all hosts are allowed to use DNS but only a subset of them is allowed to use the SNMP services provided by the SP. Currently most access providers only manage their own equipment. They assume that the customer has his own arrangements with the rest of the network, the service providers. In this generic application scenario active packet technology is used to manage packet filters across networks that belong to different SPs.

There are three road warrior laptops. Each wants a different type of network access from the other. The road warriors access the private IP network using a WaveLAN access point. These access points generate SNMP traps when a new host has been assigned an IP address. The IP address for a host is allocated by a DHCP server.

The DHCP server operates on a network controller host. This host also supports an SNMP trap collector and an injector. The injector is a process that can be used to inject active packets into the network. The router of the access network performs packet filtering and conditioning. It has an interface to the backbone network, which has two routers for egress and ingress. The former performs network address translation of packets with private IP source addresses to the address of the public interface of the egress router. The ingress router performs NAT for packets from elsewhere to services hosted in the private network.

The active packet is injected and reconfigures all the routers to accept packets for different services from the road warriors on the access network. The active packet could reconfigure all of the routers if need be.

The value added by using active packets for this scenario is that the network elements can be controlled on demand. Normally, network administrators would put a static configuration in place.

11.7 SECURITY SCENARIO

Showing a pure security scenario immediately results in quite artificial settings, especially since security is an integral part of the FAIN architecture. These are the reasons why the security scenario has been hold very general. As such it may easily become a component of any of the above presented generic application scenarios.

The security 'scenario' shows how an active packet is passed through a node and how it triggers security concepts. The scenario is applicable to any active packet approach in transport, control or management plain. The scenario consists of the following steps, which are repeated on every network node that the packet traverses:

- The DeMUX intercepts the packet and invokes security receive check function with the ANEP packet, UDP protocol information data and with the local information (service), where the packet is headed to.
- The ANEP packet is parsed.
- The hop integrity option is evaluated:
  - The correct Security Association (SA) is chosen.
  - The packet hop replay information is validated.
  - The integrity token is verified.
- If the packet contains one or more credential options:
  - The principal credentials are looked up in local cache.
  - If they are not already there, they are fetched from the previous node.
  - The credential path is validated.
  - The digital signature of the static part of the packet is verified with the trusted public key of the principal.
  - The credential option time frame is checked.
- If the packet contains active code and verification is required (e.g. due to a policy), the code is verified.
- The packet security context(s) is/are build from the principal credentials and results of the packet code verification.
- The security context of the packet is compared to the security context of the packet destination. If the packet destination (service) has defined a policy the security context(s) is/are used to authorize the access to the service.
- If the access is authorised the packet is returned to the DeMUX,
- The DeMUX passes the packet to the service.
- The packet data (variable and payload, code or data) is evaluated in the service.
- If the evaluation results in an action regarding the node, this action is authorised and the policy, if one exists, is enforced.
- The packet is returned to the DeMUX and the security check function is invoked.
- The active packet is build.
- The next hop integrity option is built:
  - Regarding packet next hop destination the right SA is chosen.
  - Replay protection value is added.
  - An integrity token is built and the hop integrity option is added to the packet.
- If every thing went successfully, the packet is returned to the DeMUX and in order to be sent to the next hop.

Any network topology with three or more active nodes will be sufficient. Requirements are the installation of the basic FAIN node services, i.e. of management, DeMUX and security facilities. Credentials and related key pairs have to be generated, SAs between the nodes has to be established before the scenario. If the policies are used they have to be supplied during the service setup or set by the network management framework.

12 CONCLUDING REMARKS

The functionality deployed in the FAIN testbed and demonstrations highlights the key achievements in the project so far:

- *Development and integration of the FAIN Active Routers and FAIN Management Systems.*
- *Installation and scenario-based runs of the FAIN testbed*

The FAIN Active Routers are deployed in the FAIN testbed as active nodes, which provide flexibility to the user for network management and service provisioning. The defining characteristic of the FAIN active router is the ability for users to load and manage software components dynamically and efficiently.

The FAIN Active Router is one of the first prototypes, which have a clear separation of the data, control and management planes running multiple types of EEs.

The FAIN management system is also one of the first prototypes of network management systems oriented to the full management of active networks.

The *FAIN Active Router* consists of the following facilities:

- *NodeOS* - an operating system for active nodes, which includes facilities for setting up and management of communications channels.
- *Resource Control Facilities* (RCF). Through resource control, resource partitioning is provided and Virtual Environments (VEs) are guaranteed that consumption stays within the agreed contract during an admission control phase, whether static or dynamic.
- *Security Facilities* (SF). The main security aspects are authentication and authorisation to control the use of resources and other objects of the node such as interfaces and directories.
- *Demultiplexing facilities* (DEMUX). As flows of packets arrive at the node, Demultiplexing filters, classifies and diverts active packets to the appropriate VE, and consequently to the destination service inside the VE.
- *Node Management Facilities* (NM). The main aspects are the initiation and maintenance of VEs, control and management of the RCF and SF, management of the mapping of external to node policies into node resource and security configurations.
- *Application/Service code deployment facilities* (ASP support). As flexibility is one of the requirements for programmable networks, partly realised as static or dynamic service provisioning, the NodeOS supports code deployment.
- *Execution Environments* (EE) are environments where application code is executed. A privileged EE manages and controls the active node and it provides the environment where network policies are executed. Multiple and different types of EE are deployed in FAIN Active Router. EEs are grouped into virtual environments (VEs), where services can be found and interact with each other.

Our demonstration on the testbed exhibits a level of integration to achieve an AN router in terms of functionality, where a node operating system, complete with integrity checks, resource monitoring facility at the active node, and an execution environment for dynamic service provisioning is proven to be running.

The *FAIN Management Station* consists of the following facilities:

- *Policies:* rules required for the management of the active nodes and network

- *Management components:* A set of mechanisms to enable network administrators to manage the active networks as a whole, including network policies set-up and processing.

Our demonstration on the testbed represents one of the earliest efforts in managing active network routers using a policy-based approach. Our management system showed high *flexibility* being able to adapt to new functionality on run-time, thus only downloading new PDP/PEP pairs when needed. Additionally, it enabled the distribution of policies via active packets. The management system is shown to provide an *efficient* framework for the management of active networks since it supported its main requirement: customisability and support for delegation.

In summary, the main concepts shown in the FAIN testbed demonstrations are:

- Delegation of management functionality
- Creation of an Active Virtual Network
- Dynamic downloading of service-specific management components
- Dynamic extensibility of the management stations functionality downloading new PDPs when they are needed
- Dynamic installation of an Active Service within the active router in the control and data planes.
- Multiple Execution Environments (EEs) grouped and managed as Virtual Environments (VEs)
- Clear separation of data, control and management planes
- Hop-by-hop authentication and integrity checks of packets before delivering them to the VE
- Run-time configuration of the demultiplexing/dispatching and multiplexing functionality. The Demultiplexing component dispatches packets to the configured services.

Today, there have been assertions and proposals within the research community about finding the 'killer application' that can be created by Active Networking (AN) technology. In our demonstration, we note that AN technology should be viewed as a means to an end and not the end game itself. This can be seen from the service deployment scenario, which is dynamic process initiated by the user. In particular we show how AN can provide a novel approach to instantiate the Transcoder application. We also illustrated an active shaper service, which is used to limit the cross-traffic entering an AVPN link, as well as the creation of active channels within the node that redirect flows to active service running within a particular VE.

One of the salient features in our testbed included the instantiation of a VE on an AN node. This further enabled the introduction of management enforcement points within the node VEs to reduce management traffic. In our implementation, the SP obtained a restricted delegation of management functionality to manage its resources. It was even able to do so with its proprietary code. Hence, we feel that the *delegation* approach was proven to be adaptable to different business models and user needs.

The PDP Manager component in collaboration with the FAIN ASP system dynamically detected the needed functionality. It downloaded it and installed it on run-time, hence extending the functionality supported by the management framework. The SP in the use case was able to manage its allocated resources, and tailor them to the needs of the implemented service. It was even able to *customise* the management framework for its needs with its own management code. Additionally, at the element-level, PEPs that run within the active node enhance the distribution of management functionalities over the network. This property aims to solve *scalability* problems inherent in centralised policy-based management architectures.

The *FAIN testbed* consists of the following facilities:

I. FAIN Active Routers:

Components' Interoperability: the components developed by different partners are interoperable, functionality has being specified via a common method and components are using common protocols for exchanging information;

Execution Environments Integration: integration of implementations in different execution environments EEs;

Components' Portability: the components are designed to run on different platforms: PCs and Hybrid Active Router, which combine a commercial router with an active network EE provided by a physically separate and attached PC.

Uniform Node management: the inventory of components, e.g., active services, RCF components and the security component are managed in a uniform way;

Access control: the access to components is conducted in a restricted manner in order to avoid policies enforcement by malicious parties;

Usage monitoring: the monitoring system ensures that the enforced policies adhere to the expected outcome;

Components' Deployment: the deployment of components are carried out automatically with resolution amongst dependencies;

Configuration*:* support for configuring components during the set-up and dynamic re-configuration during runtime is required in order to tailor applications as per user;

Dynamic update*:* there is also support for dynamically update components – plug-and-play, unplug-and-play.

II. FAIN Management Stations:

High flexibility being able to adapt to new functionality at run-time by downloading new PDP/PEP pairs only when needed. Additionally, it enabled the distribution of policies via active packets.

Delegation approach is proven to be secure and highly adaptable to different business models and user needs. Users are able to obtain a restricted delegation of management functionality with which they are able to manage their resources, even with their service-specific code.

Extending the functionality support by the management system The PDP Manager component in collaboration with the ASP system dynamically detects the needed functionality.

Customisation of the management framework. A user is able to manage its allocated resources, and tailor them to the needs of its implemented service.

Highly distributed FAIN management framework, with several element level management stations and one network level management station constituting the management infrastructure. Additionally, at the element level PEPs run within the active node distributing even more the management functionality over the network.

Scalability: two-tier policy-based management architecture for support of scalability of distributing policies to the management stations where they can be best processed

Efficient framework for the management of active networks since it supports its main requirements, i.e., dynamic extensibility, customisability and delegation support.

13 FUTURE DEVELOPMENTS IN PROGRAMMABLE SERVICE NETWORKS

This section identifies some of the research and development issues, which are underpinning the migration from programmable networks towards programmable service networks.

13.1 REFERENCE ARCHITECTURE FOR PROGRAMMABLE SERVICE NETWORS

Since its beginning, the Internet's development has been founded on a basic architectural premise: a simple network service is used as a universal means to interconnect intelligent end-systems. The end-to-end argument has served to maintain this simplicity, by pushing complexity into the endpoints, which require it, allowing the Internet to reach an impressive scale in terms of inter-connected devices.

However, while the scale has not yet reached its limits, the growth in functionality - the ability of the Internet to adapt to new functional requirements - has slowed with time. We see new network architectures evolving both above and below the Internet. Underneath the Internet, all-optical networks lacking buffering will dominate the core, while wireless LANs subject to high packet error rates as well as ADSL access networks will dominate the edges. Above the Internet are systems, which exploit and demand mobility, the ability to deliver continuous media, private address spaces and access control. While some of the requirements were met quickly with ad-hoc solutions and de-facto standards, others received painful, long-lasting, and still unsuccessful attention: QoS, IPv6 and a well-designed multicast service, despite having standards, are still not deployed.

Another unfortunate consequence of the existing architecture is the division between network engineers adopting the viewpoint of protocols and software engineers who advocate a larger use of object oriented technologies and methodologies in engineering the network.

To complicate even further the problem space, the responsibility of managing and controlling the end-to-end path is fragmented into different administrative domains, providers who own parts of the network applying different business models, marketing policies and using a variety of technologies, as mentioned above. Accordingly, delivering an application to the end user (the customer) is not a trivial task, as it requires a complex coordination among these different players over a heterogeneous network.

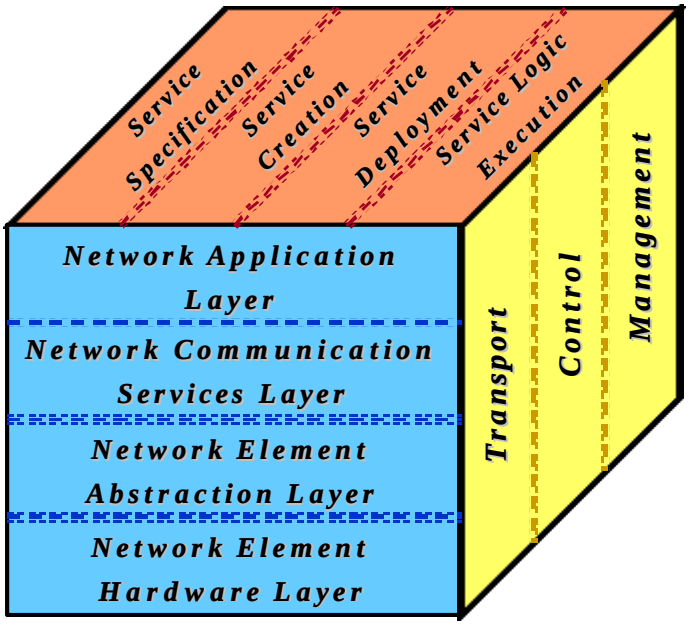


Figure 11.1: Conceptual & Reference Model

We strongly believe that the next generation network, called ***Polymorphic Internet***, must be built on a middleware engineered in such a way that allows all the different parties to coexist without impeding future evolution. The key to this is the rapid application/service creation and deployment, as these are the true commodities that must be promoted by providers and purchased by customers thereby creating a vast fuel tank that will constantly supply the network expansion. In order to achieve this, it is essential that we bring together the two communities of engineers and foster a networking paradigm where they both contribute in a synergistic way.

In addition to the technological drivers such as link technologies and application software, new and interesting technologies have appeared in the realm of network architecture, and we believe that they can be applied to a new discipline of networking, in which the end-to-end argument's logic is still maintained, but avoiding an overly simplistic and rigid interpretation as in today's Internet. The innovations in network architecture include:

- Open & Programmable networks
- Active networks
- Network processors intended for the next generation of routers
- Open and extensible software interfaces
- Component models and architectures
- Service modelling and description
- Dynamic Service creation and deployment
- Autonomic Service Activation
- Active Middleware Service
- Network and Service Management
- Peer-to-peer networks and services
- Multimedia and Audio-visual services and systems

A common conceptual model unites the most advanced work in each of these areas, which is captured in the reference model shown above in Figure , which succinctly defines the scope of our proposed activities.

Our Reference Model expands across three dimensions that synergistically make up of the whole Polymorphic Internet environment.

More specifically, the first dimension (front side of the cube) argues for a layered architecture comprising four main layers with a specific scope assigned to each. The functionality provided by each layer is exposed using open interfaces. Components running in any layer may access additional functionality available at the lower layer in a seamless way. To this end, services and their logic may be distributed across different layers or may be confined within only one layer. Thus, for example, in the model we proposed of an all-optical core, the core might be source-routed if packet-switched, or circuit switched. The network communication services would react to the particular design choice made to cope with the current limitations of photonics, and similar architectural choices might be made for shared wireless links at the edge (incidentally we expect the great majority of network enhancement software will operate on network elements lying between the wireless mobile edge and the stationary all-optical core).

The choice of these particular layers and the scope thereof is the result of converging conclusions collected from state-of-the-art forums, such as IEEE P1520, IETF ForCES, MSF, JAIN, and OSA, and by the FAIN experience.

The bottom two layers, namely, the Network Element Hardware Layer, and the Network Element Abstraction Layer, refer to the large family of Network Elements (NE) or components thereof, like Routers, Optical Switches, and Gateways etc. found in abundance in every part of the Network. These NEs participate in forwarding and processing flows that belong to specific services and users' thereof. To this end, the scope of these layers is about the functionality provided by the NE. The behaviour of these NEs is influenced by means of open interfaces that abstract the NE functionality in the NE Abstraction Layer.

In contrast, the upper two layers, the Network Communication Services Layer and the Network Application Layer, break down the functionality of network-deployed services into the communication part and application part, respectively.

The former deals with how communication is established by distributed entities of a service. The type of services found in this layer are standard services that enable the transfer of packets in a network, and advanced support communication services, which provide certain extra features and support to service and application providers, according to their requirements. In contrast, the latter (the Network Application Layer) is user specific and deals with end-to-end issues.

In this way we decouple the communication aspect from an end-user application, which becomes accessible by means of the interface defined between the two layers.

The second dimension, (top side of the cube) refers to the concept of Service Creation and Execution Environments (SC&EE) that are going to be deployed across the layers above and host and run these services. We identify four main conceptual areas for developing an SC&EE: Service Specification Service Creation, Service Deployment, and Service Logic Execution. As part of the first area, a service must be able to be specified in an unambiguous way that not only identifies the service components that collectively define the service but also where these components should be deployed. In contrast, Service Creation addresses issues of creating complex services from existing, simpler services. Consequently, Service Deployment deals with the way that the service and its components are deployed. Again, decoupling the specification from the creation and deployment mechanism allows work in these areas to progress independently, offering their functionality by means of open interfaces. Finally, the service logic defines the operations and management of a service component and system. Such operations may access the open interfaces of the first dimension since they must be visible inside the SC&EE or exchange messages with other components not necessarily part of the same service. We also believe that a common service specification and message-driven communication mechanism will greatly facilitate integration of network services contributed and deployed by different parties on different platforms and heterogeneous networks. Protocols like SOAP as well as new XML-based languages are deemed as essential achieving this goal.

Finally, the two dimensions, namely, the architectural layers and the SC&EE may span all three operational planes - management, control and transport - where we argue for a clear separation among them. For instance, SC&EE could be designed and built exclusively for one of the operational planes, by taking into account the requirements, limitations, and suitable available technology.

We note here that the reference model of Figure may also imply a new business model. By creating these layers and conceptual areas we intend to unleash new innovations and resulting market forces, promoting healthy competition and stimulating new products for specific areas of the model, either alone or jointly. As a consequence, new products or prototypes in the form of hardware, middleware, algorithms, designs, frameworks, services, applications, interface specifications, software, demonstration, testbed, etc will be developed and integrated giving flesh and bones to the reference model of Figure .

We aspire to develop a new discipline of service network programming, focussed on a network, namely, the Polymorphic Internet, which would enable users or service providers to create their own or extend and configure existing service networks.

13.2 REQUIREMENTS ANALYSIS FOR FURTHER DEVELOPMENT IN PROGRAMMABLE SERVICE NETWORKS

There is a striking discord between the great intellectual successes of research and advanced development in networking, computing and other related technologies, and the challenging financial environment in the industry in spite of huge investment. How can this be? We argue here that the problem is caused by too great a concentration on the technologies themselves with insufficient attention paid to the needs of the paying customer, and the business needs of the providers. We need to change the customers' perception of the Network[5] by improving the quality of its support for the services they want to use, customised and controlled by themselves and are willing to pay for, while a maintaining strong awareness of the broader business issues. We are envisaging such a project, the Polymorphic Internet, which is targeted at exactly this issue: the rapid, easy, and cost effective introduction of new value-added services and the customised configuration of the networks to support them. This implies that:

- the shared network resources should be tailored specifically for each application's requirements, doing away with the current "lowest common denominator" approach
- the network elements should be programmable in order to allow the provision of the necessary service flexibility in networks and access to this programmability should be provided to users

There is a need to identify and fully understand the issues involved in this transformation, whether evolutionary or revolutionary. Unless these issues are addressed, the networking industry will remain mired in its financial tar pit, eventually quashing innovation as even resources for R&D become scarce. The following are key factors, which need to be addressed.

- **Network Heterogeneity**. The Network is heterogeneous, both in components and in administrative policy. What is perceived by the end user as a single "network" is in fact composed of networks of different technologies (access, edge, core etc), administered by different authorities and owned by a variety of providers (e.g. Access Providers, ISPs, Content Providers, Application Providers) with diverse or competing business models and strategies.

- **Commodity.** The Return on Investment (RoI) when introducing new access technologies such as ADSL relies on selling useful value added services to customers. Bandwidth alone requires significant capital investment but is easily commoditised, and offers relatively low returns. There is therefore a pressing need to ensure that innovative commercial applications can be effectively supported by public networks.

- **Inflexible Service Deployment.** Attempts to introduce and rapidly deploy new applications or value-added services are hampered by Network heterogeneity and lack of configurability, and so current deployment is restricted to homogeneous subnetworks which in turn limits the customer base.

[5] Throughout this document the use of the term "Network" is used to declare the customer's end-to-end path experience comprised of a number of different networks, telecommunication networks, data networks or otherwise.

The Polymorphic Internet would contribute to the pervasive service vision by making networks service driven and enabled. It will also contribute to a new view of networking: from a generic commoditised "pipe" for bit transport to a new, richer platform, which can support services in a newly enabled market.

13.3 EXPECTED KEY NOVEL FEATURES

Next Generation Networks (NGN) are heterogeneous and multipurpose communication networks. Polymorphic Internet (***p***) is an NGN service network, which fully supports Ubiquitous Computing and Communications. In this context Polymorphic Internet key novel features are:

- ***p*** is an IP based network, which is capable of dynamic reconfiguration and autonomic adaptation to service requirements, user activity and network conditions. However, reconfiguration and adaptation is not restricted to IP but may span all network layers if necessary.
- ***p*** networks have a high degree of autonomy in that they are reconfigured and adapted on demand by autonomous service and network management components.
- ***p*** networks provide resources to network users, applications and operators (communication, storage, computing and content resources) at any place and any time.
- ***p*** networks provide support for ubiquitous applications e.g. provision of storage and computing resources, mobility and security in a very flexible and customisable way(Ubiquitous Computing).
- ***p*** is a set of cooperative service specific network overlays of physical and logical resources - ***p*** are nested cooperative networking nodes comprising highly dynamic middleware environments that provide value added services over a range of wired and wireless network technologies (Ubiquitous Communications).
- ***p*** networks offer application service providers a set of interfaces and tool kits that facilitate flexible creation, rapid deployment and (re)configuration/personalisation of new services.
- ***p*** networks offer the user the opportunity to customise the creation and participate in the management of adaptable connectivity services and audio-visual services.

The following diagram summarises the key and novel characteristics of the ***p*** networks.

	p Programmability	p Autonomy	p e2e View
	Network modification	Enablers for modification	Dynamic Enablers & net modification
Results	Reconfigurability	Autonomic reconfiguration	Dynamic reconfiguration
Target	Net modification	Node modification	Net & Node Dynamic modification
Systems	Peer layers/overlays	Top Down Layered Overlays	Top Down & Vertical Layered Overlays
Initiated by	Net requirements for change	Service requirements for change	Business requirements for change
Inter - operability	Network Components	Node components	Service, Net, Node Components
Driven by	Local/global system optimisation	Local / global self - organisation	Business organisation

Figure 11.2 Key Characteristics of Polymorphic Internet

14 REFERENCES FOR STATE OF THE ART

[2] Schwartz, B., et al., "Smart Packets for Active Networks", http://www.net-tech.bbn.com/smtpkts/smart.ps.gz, January 1998

[3] Wetherall, D., Tennenhouse, D., "The ACTIVE IP Options", In Proceedings of the 7th ACM SIGOPS European Workshop, September 1996

[4] Wetherall, D., et al., "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", In Proceedings of IEEE OPENARCH'98, April 1998

[5] Decasper, D., Dittia, Z., Parulkar, G., Plattner, B., "Router Plugins: A Software Architecture for Next Generation Routers", In Proceedings of ACM SIGCOMM'98, Vancouver, Canada, September 1998

[6] Thompson, K., Miller, G., Wilder, R., "Wide-Area Internet Traffic Patterns and Characteristics", In IEEE Network, November/December 1997

[7] FAIN Deliverable 1 "Requirements Analysis and Overall AN Architecture", May 2001 –http://www.ist-fain.org

[8] NetBSD, http://www.netbsd.org

[9] Linux, http://www.kernel.org

[10] St. Schmid, T. Chart, M. Sifalakis, A.C. Scott, Flexible, Dynamic, and Scalable Service Composition for Active Routers, In Proceedings Fourth Annual International Working Conference on Active Networks (IWAN 2002), Zürich, Switzerland, Lecture Notes in Computer Science 2546, Springer Verlag, Berlin Heidelberg New York, December, 2002.

[11] Architectural Framework for Active Networks, Draft version 1.0, K.L. Calvert, ed., July 27, 1999. http://protocols.netlab.uky.edu/~calvert/arch-latest.ps

[12] DARPA Active Network Program, 1996, http://www.darpa.mil/ato/programs/activenetworks/actnet.htm

[13] Open Signalling Working Group, http://www.comet.columbia.edu/opensig/.

[14] Biswas, J., et al., "The IEEE P1520 Standards Initiative for Programmable Network Interfaces", IEEE Communications, Special Issue on Programmable Networks, Vol. 36, No 10, October, 1998. http://www.ieee-pin.org/

[15] Khosravi, H., and T. Anderson (eds.), "Requirements for Separation of IP Control and Forwarding", January 2003, http://www.ietf.org/internet-drafts/draft-ietf-forces-requirements-08.txt

[16] Bjorkman, N., Y. Jiang, et. al., "The Movement from Monoliths to Component-Based Network Elements", IEEE Communications, Special Issue on Telecommunications Networking at the Start of the 21st Century, Vol. 39, No. 1, January 2001. http://www.msforum.org/techinfo/IEEEcomMag200101_monoliths.pdf

[17] Campbell, A. T., H. De Meer, M. E. Kounavis, K. Miki, J. Vicente, and D. Villela "A Survey of Programmable Networks", ACM Computer Communications Review, April 1999, http://www.acm.org/sigcomm/ccr/archive/1999/apr99/ccr-9904-campbell.pdf

[18] RFC 2475, "An Architecture for Differentiated Services", 1998. http://www.ietf.org/rfc/rfc2475.txt

[19] Intel, IXP family of network processors, http://www.intel.com/design/network/products/npfamily/

[20] IBM Network Processors, http://www-3.ibm.com/chips/products/wired/products/network_processors.html

[21] "Initial Active Network and Active Node Architecture", FAIN Project Deliverable 2, http://www.ist-fain.org/deliverables/del2/d2.pdf

[22] IETF ForCES, http://www.ietf.org/html.charters/forces-charter.html

[23] IEEE P1520.2, Draft 2.2, Standard for Application Programming Interfaces for ATM Networks, http://www.ieee-pin.org/pin-atm/intro.html

[24] Biswas et al., "Proposal for IP L-interface Architecture", IEEE P1520.3, P1520/TS/IP013, 2000. http://www.ieee-pin.org/doc/draft_docs/IP/p1520tsip013.pdf

[25] Vicente, J., S. Denazis, et al., "L-interface Building Block APIs", IEEE P1520.3, P1520.3TSIP016, 2001. http://www.ieee-pin.org/doc/draft_docs/IP/P1520_3_TSIP-016.doc

[26] Vicente, J., M. Kounavis, D. Villela, M. Lerner, and A. Campbell, "Programming Internet Quality of Service", 3rd IFIP/GI International Conference of Trends towards a Universal Service Market, Munich, Germany, September 12-14, 2000. http://comet.ctr.columbia.edu/~campbell/papers/usm00.pdf

[27] IETF ForCES, draft-ietf-forces-framework-04.txt, December 2002, http://www.ietf.org/internet-drafts/draft-ietf-forces-framework-04.txt

[28] Yang, L., J. Halpern, R. Gopal, R. Dantu, "ForCES Forwarding Element Functional Model", March 2003.

[29] D. Scott Alexander, William A. Arbaugh, Michael W. Hicks, Pankaj Kakkar, Angelos D. Keromytis, Jonathan T. Moore, Carl A. Gunter, Scott M. Nettles, and Jonathan M. Smith "The SwitchWare Active Network Architecture", IEEE Network Special Issue on Active and Controllable Networks, vol. 12 no. 3, pp. 29 – 36, 1998. http://www.cis.upenn.edu/~switchware/papers/switchware.ps

[30] "Node OS Interface Specification", AN Node OS Working Group, Larry Peterson, ed., November 30, 2001. http://www.cs.princeton.edu/nsg/papers/nodeos-02.ps

[31] K. Calvert, S. Bhattacharjee, E. Zegura and J. Sterbenz, "Directions in Active Networks", IEEE Communications Magazine, 1998. http://www.cc.gatech.edu/projects/CANEs/papers/Comm-Mag-98.pdf

[32] Johnathan M. Smith, Kenneth Calvert, Sandy Murphy, Hilarie K. Orman, and Larry L. Peterson, "Activating Networks: A Progress Report", IEEE Computer 32(4):32–41, April 1999. http://www.cs.princeton.edu/nsg/papers/an.ps

[33] Steven Berson, Bob Braden, and Livio Ricciulli, "Introduction To The ABone", June 15, 2000. http://www.isi.edu/abone/DOCUMENTS/ABoneIntro.ps

[34] J.E. van der Merwe, S. Rooney, I.M. Leslie and S.A. Crosby, "The Tempest - A Practical Framework for Network Programmability", IEEE Network, Vol 12, Number 3, May/June 1998, pp.20-28. http://www.research.att.com/~kobus/docs/tempest_small.ps

[35] Bhattacharjee, S., "Active networks: Architectures, Composition, and Applications", Ph.D. Thesis, Georgia Tech, July 1999

[36] B. Braden, A. Cerpa, T. Faber, B. Lindell, G. Phillips, J. Kann and V. Shenoy, "Introduction to the ASP Execution Environment (Release 1.5)", Nov 30, 2001. http://www.isi.edu/active-signal/ARP/DOCUMENTS/ASP_EE.ps

[37] B. Braden, T. Faber, M. Handley, "From Protocol Stack to Protocol Heap – Role Based Architecture", HotNets I, Princeton University, USA, October 2002, http://www.cs.washington.edu/hotnets/papers/braden.pdf

[38] Alden W. Jackson, James P.G. Sterbenz, Matthew N. Condell, Joel Levin, David J. Waitzman, "The SENCOMM Architecture", Technical Report, BBN Technologies, 26 April 2000. http://www.ir.bbn.com/projects/sencomm/doc/architecture.ps

[39] M. E. Kounavis, A. T. Campbell, S. Chou, F. Modoux, J. Vicente, and H. Zhang, "The Genesis Kernel: A Programming System for Spawning Network Architectures", IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Active and Programmable Networks, Vol. 19, No. 3, pp. 49-73, March, 2001. http://comet.ctr.columbia.edu/genesis/papers/jsac2001.pdf

[40] David Wetherall, "Service Introduction in an Active Network", Ph.D. Thesis, available as MIT/LCS/TR-773, February 1999.

[41] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles, "PLAN: A Packet Language for Active Networks", in Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages, pages 86-93. ACM, September 1998. http://www.cis.upenn.edu/~switchware/papers/plan.ps

[42] Pankaj Kakkar, Michael Hicks, Jonathan T. Moore, and Carl A. Gunter, "Specifying the PLAN networking programming language" in Higher Order Operational Techniques in Semantics, volume 26 of Electronic Notes in Theoretical Computer Science. Elsevier, September 1999. http://www.cis.upenn.edu/~switchware/PLAN/spec/spec-experience.ps

[43] SNAP: Safe and Nimble Active Packets, http://www.cis.upenn.edu/~dsl/SNAP/

[44] Michael Hicks, Jonathan T. Moore, and Scott Nettles, "Compiling PLAN to SNAP", IWAN'01, September/October 2001. http://www.cis.upenn.edu/~jonm/papers/plan2snap.ps

[45] Composable Active Network Elements Project (CANES). http://www.cc.gatech.edu/projects/canes/

[46] S. Merugu, S. Bhattacharjee, E. Zegura and K. Calvert, "Bowman: A Node OS for Active Networks", Proceedings of IEEE Infocom 2000, Tel Aviv, Israel, March 2000. http://www.cc.gatech.edu/projects/CANEs/papers/bowman.pdf

[47] "Composable Services for Active Networks", AN Composable Services Working Group, Ellen Zegura, editor, September 1998. http://www.cc.gatech.edu/projects/CANEs/papers/cs-draft0-3.ps.gz

[48] S. Berson, B. Braden and E. Gradman, "The Network I/O daemon – netiod", October 11, 2001, Draft version. http://www.isi.edu/abone/DOCUMENTS/netiod.ps

[49] "Revised Active Network and Active Node Architecture", FAIN Project Deliverable 4, http://www.ist-fain.org/deliverables/del4/d4.pdf.

[50] ABONE Testbed, http://www.isi.edu/abone/

[51] Internet Engineering Task Force – http://www.ietf.org

[52] Distributed Management Task Force – http://www.dmtf.org

[53] Common Information Model Standards - http://www.dmtf.org/standards/standard_cim.php

[54] B. Moore, Policy Core Information Model (PCIM) Extensions, RCF3460, January 2003

[55] Resource Allocation Protocol IETF's WG - http://www.ietf.org/html.charters/rap-charter.html

[56] N. Damianou, N. Dulay, E. Lupu, M Sloman, The Ponder Specification Language, Workshop on Policies for Distributed Systems and Networks (Policy2001), HP Labs Bristol, 29-31 Jan 2001 - http://www.doc.ic.ac.uk/~mss/Papers/Ponder-Policy01V5.pdf

[57] The Jasmin Project - http://www.ibr.cs.tu-bs.de/projects/jasmin/policy.html

[58] Seraphim Project homepage, Seraphim: Building Dynamic Interoperable Security Architecture For Active Networks, http://choices.cs.uiuc.edu/Security/seraphim/

[59] Active Network Distributed Open Infrastructure Development (ANDROID) - http://www.cs.ucl.ac.uk/research/android/

[60] Application Level Active Networks - http://dmir.it.uts.edu.au/projects/alan/

[61] Y. Kanada, Dynamically extensible policy server and agent, Policies for Distributed Systems and Networks, 2002. Proceedings, pages 236-239

[62] M. Fonseca, N. Agoulmine, O. Cherkaoui, Active Networks as a Flexible Approach to deploy QoS Policy Based Management, http://citeseer.nj.nec.com/483138.html

[63] K. Kato, S. Shiba, Designing Policy Networking System Using Active Networks, Second International Working Conference on Active Networks (IWAN'2000), Tokyo, Japan, October 2000, Springer-Verlag LNCS, Yasuda (ed.).

[64] M. Sloman, and E. Lupu, Policy Specification for Programmable Networks, International Working Conference on Active Networks (IWAN'99), Berlin, Germany, June-July 1999, Springer-Verlag LNCS, Stefan Covaci (ed.).

[65] Stevenson, D. W., 'Network management: what it is and what it isn't', White Paper, April 1995, (http://netman.cit.buffalo.edu/Doc/DStevenson/)

[66] Sloman, M. (ed.), 'Network and distributed systems management', Addison-Wesley, 1994

[67] Galtier, V., Mills, K. L., Carlinet, Y., Bush, S. F., Kulkarni, A., 'Prediction and controlling resource usage in a heterogeneous Active Network', MILCOM 2001, McLean, October 2001

[68] Hood, C. S., Ji, C. 'ProActive Network fault detection', INFOCOM '97, Kobe, April 1997

[69] Goldszmidt, G., Yemini, Y., 'Distributed management by delegation', 15th International Conference on Distributed Computing Systems, Vancouver, June 1995

[70] Sugauchi, K., Miyazaki, S., Covaci, S., Zhang, T., 'Efficiency evaluation of mobile agent –based network management system', p. 527, LNCS, April 1999

[71] Kawamura, R., Stadler, R., 'Active distributed management for IP networks', IEEE Communications Magazine, pp. 114-120, April 2000

[72] Greenwood, D., Gavalas, D., 'Using active processes as the basis for an integrated distributed network management architecture', 1st International Working Conference on Active Networks (IWAN '99), Berlin, June 1999

[73] The Open Group. Inter-Domain Management: Specification Translation & Interaction Translation. Technical Standard C802, January 2000

[74] Matthias Bossardt, Takashi Egawa, Hideki Otsuki, Bernhard Plattner: Integrated Service Deployment for Active Networks. In Proceedings Fourth Annual International Working Conference in Active Networks (IWAN 2002), Zürich, Switzerland, Lecture Notes on Computer Science 2546, Springer Verlag, Berlin Heidelberg New York, December, 2002.

[75] TBD – Reference to Content Delivery Network

[76] Decasper, D., Parulkar, G., Choi, S., DeHart, J., Wolf, T., Plattner, B., "A Scalable, High Performance Active Network Node", In *IEEE Network*, January/February 1999

[77] Michael Fry and Atanu Ghosh, Application Level Active Networking, Computer Networks, 31 (7) (1999) pp. 655-667

[78] B.Mathieu and al. "Deployment of Services into Active Networks" In Proceedings WTC-ISS 2002, Paris, France, September 2002

[79] Open Service Gateway Initiative - www.osgi.org

[80] OSGI Service Platform, Release 2, October 2002

[81] Object Management Group - www.omg.org

[82] CORBA Components, v3.0 full specification: Document -formal/02-06-65.

[83] Marcin Solarski, Matthias Bossardt, Thomas Becker: Component-based Deployment and Management of Services in Active Networks. In Proceedings Fourth Annual International Working Conference on Active Networks (IWAN 2002), Zürich, Switzerland, Lecture Notes in Computer Science 2546, Springer Verlag, Berlin Heidelberg New York, December, 2002.

[84] Gislj Hjalmtysson, "The Pronto Platform – A Flexible Toolkit for Programming Networks using a Commodity Operating System, OpenARCH 2000.

[85] Andy Bavier, Thiemo Voigt, Mike Wawrzoniak, Larry Peterson, Per Gunningberg, SILK: Scout Paths in the Linux Kernel.

[86] A.Montz, D. Mosberger, S. O'Malley, L. Peterson, T. Proebsting. Scout: A Communications-Oriented Operating System. IEEE HotOS Workshop (May 1995)

[87] Ralph Keller, Lukas Ruf, Amir Guindehi, Bernhard Plattner, PromethOS: A Dynamically Extensible Router Architecture Supporting Explicit Routing, Proceedings of the Fourth Annual International Working Conference on Active Networks (IWAN 2002), Springer Verlag, Lecture Notes in Computer Science, 2546, December 4.-6. December 2002, Zurich, Switzerland.

[88] Ian Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul Barham, David Evers, Robin Fairbairns, and Eoin Hyden, The Design and Implementation of an Operating System to Support Distributed Multimedia Applications, http://www.cl.cam.ac.uk/Research/SRG/netos/old-projects/nemesis/documentation.html

[89] Dawson R. Engler, M. Frans Kaashoek, James O'Toole Jr, Exokernel: an operating system architecture for application-level resource management, In the Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95), Copper Mountain Resort, Colorado, December 1995, pages 251-266

[90] Moab, http://www.cs.utah.edu/flux/janos/moab.html

[91] Patrick Tullmann, Mike Hibler, and Jay Lepreau. Janos: A Java-oriented OS for Active Networks. IEEE Journal on Selected Areas of Communication. Volume 19, Number 3, March 2001.

15 OTHER REFERENCES

[1] S. Aidarous and T. Pevyak (eds), "Telecommunications Network Management: Technologies and Implementations", IEEE Press, 1997.

[2] J.P. Martin-Flatin, S. Znaty. "A Simple Typology of Distributed Network Management Paradigms", Proc. Of 8th IFIP/IEEE International Workshop on Distributed Systems, Operations & Management (DSOM '97), Sydney, October 1997.

[3] M. Kahani, and H.W.P. Beadle, "Decentralised Approaches for Network Management", Comp. Comm. Review, Vol. 27, No. 3, July 1997.

[4] D. Drogseth, "The New Management Landscape", PACKET, CISCO System Users Magazine, Third Quarter, 2002

[5] N. Damianou, N. Dulay, E. Lupu, M Sloman, "The Ponder Specification Language" Workshop on Policies for Distributed Systems and Networks (Policy2001), HP Labs Bristol, 29-31 Jan 2001 - http://www.doc.ic.ac.uk/~mss/Papers/Ponder-Policy01V5.pdf

[6] R.Yavatkar, D.Pendarakis, R.Guerin "A framework for Policy-based Admission Control" RFC 2753, January 2000.

[7] K. Chan, et. al., "COPS Usage of Policy Provisioning", IETF RFC 3084, March 2001.

[8] D. Verma, MacMillan Technology Series, "Supporting Service Level Agreements on IP Networks", Macmillan Technical Publishing U.S.A

[9] J.E. van der Merwe, S. Rooney, I.M. Leslie and S.A. Crosby, "The Tempest - A Practical Framework for Network Programmability", IEEE Network, Vol 12, Number 3, May/June 1998, pp.20-28. http://www.research.att.com/~kobus/docs/tempest_small.ps

[10] G. Goldszmidt and Y. Yemini. "Distributed Management by Delegating Mobile Agents". In The 15th International Conference on Distributed Computing Systems, Vancouver, British Columbia, June 1995
http://www.cs.columbia.edu/~german/papers/icdcs95.ps.Z

[11] REF FAIN Deliverable D1: Requirements Analysis and Overall Architecture. FAIN Consortium, May 2001, pp. 11-18

[12] M.Solarski, M.Bossardt, T.Becker "Component-based Deployment and Management of Services in Active Networks", IWAN'02, Zürich, CH, Dec. 2002.

[13] Yang, L., J. Halpern, R. Gopal, R. Dantu, "ForCES Forwarding Element Functional Model", March 2003.

[14] J.E. van der Merwe, S. Rooney, I.M. Leslie and S.A. Crosby, "The Tempest - A Practical Framework for Network Programmability", IEEE Network, Vol 12, Number 3, May/June 1998, pp.20-28. http://www.research.att.com/~kobus/docs/tempest_small.ps

[15] M. E. Kounavis, A. T. Campbell, S. Chou, F. Modoux, J. Vicente, and H. Zhang, "The Genesis Kernel: A Programming System for Spawning Network Architectures", IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Active and Programmable Networks, Vol. 19, No. 3, pp. 49-73, March, 2001. http://comet.ctr.columbia.edu/genesis/papers/jsac2001.pdf

[16] Bhattacharjee, S., "Active networks: Architectures, Composition, and Applications", Ph.D. Thesis, Georgia Tech, July 1999

[17] B. Braden, A. Cerpa, T. Faber, B. Lindell, G. Phillips, J. Kann and V. Shenoy, "Introduction to the ASP Execution Environment (Release 1.5)", Nov 30, 2001. http://www.isi.edu/active-signal/ARP/DOCUMENTS/ASP_EE.ps

[18] "Revised Active Network and Active Node Architecture", FAIN Project Deliverable 4,

http://www.ist-fain.org/deliverables/del4/d4.pdf.

[19] "Specification of Revised Case Study Systems", FAIN Project Deliverable 5, http://www.ist-fain.org/deliverables/del5/d5.pdf.

[20] FAIN Project Deliverables – www.ist-fain.org/deliverables

- D1-Requirements Analysis & Overall AN Architecture
- D2-Initial Active Network and Active Node Architecture
- D3-Initial Specification of Case Study Systems
- D4-Revised Active Network Architecture and Design
- D5-Specification of Revised Case Study Systems
- D6-Definition of evaluation criteria and plan for the trial
- D7-Final Active Network Architecture and Design
- D8-Final Specification of Case Study Systems
- D9-Evaluation Results and Recommendations
- D40-FAIN Demonstrators and Scenarios

