Management of Active and Programmable Networks

Alex Galis[1], Alvin Tan[1], Joan Serrat[2], Julio Vivero[2]

[1] University College London, Department of Electrical and Electronic Engineering, Torrington Place, London WC1E 7JE, United Kingdom; Tel: +44 20 7679 3956, email: {a.galis, atan}@ee.ucl.ac.uk

[2] Universitat Polit cnica de Catalunya, Dept Teoria del Senyal i Comunicacions, Jordi Girona, 1-3, 08034 Barcelona, Spain; Tel: + 34 93 401 6586, email: {serrat, vivero}@tsc.upc.es

ABSTRACT

This paper reviews the characteristics and methodologies developed by the programmable network and active network communities as complementary approaches making a significant contribution to the development of new architectures. It reviews the characteristics and methods developed for policy-based network management, specifically in the context of active networks. A network management overview of existing work in active and programmable networks is given. One such architecture is developed in the IST Project FAIN [A.3], [A.20]. FAIN is a three-year project, whose main task is to develop and validate an open, flexible, programmable and dependable network architecture based on novel active node concepts. The FAIN policy based network management is described.

KEY WORDS: Active and Programmable Networks, Network Management, Policy Based Management

1. INTRODUCTION

The networking research community has realised for sometime now the need for more flexibility and dynamically customisable networks. This realisation stems from the requirement that networks should be engineered in such a way that facilitate and support the rapid creation and deployment of new services. Meeting this requirement has lead to a change of the one-dimensional networking model, traditionally captured by the communication model of packet header processing and forwarding, to the two-dimensional one with the addition of the computational model. The combination of the two is what it defines *a programmable network.*

Programmable networks [A.13] do not come cheap as they impose a number of requirements on operators, and hardware vendors while they introduce a series of technical challenges to the research community. The repercussion of the former is that a new business model emerges as vendors and operators should open-up their resources to third-party service providers and users thereby giving rise to new business roles and competition. The repercussion of the latter is that a new network architecture has to be defined based on a novel network programming model that leverages new network technologies and research advances from the fields of programming languages, operating systems, object orientation, distributed systems etc.

In search of fulfilling such objectives two schools of thoughts have now clearly emerged. The first school of thought is represented by the Opensig (Open Signalling) [A.0] community and operates through a series of international workshops whereas the second is represented by the Active Network community [A.29] and operates through a series of projects mainly under the auspices of DARPA [A.16].

Both communities have investigated and experimented with different architectural approaches, using a variety of technologies and focusing on different objectives. The Opensig community advocates that programmability can be achieved by means of defining a series of open network interfaces that represent physical network devices and network services as distributed objects. To this end, independent service providers are allowed to enter the telecommunications market and compete with large vendors and network operators by deploying their own value-added or more efficient services. These ideas have been formalised in the standardisation activity of IEEE Project 1520 [A.4], [A.21] on Programmable Interfaces for Networks.

In contrast, Active Networks community have opted for a more dynamic approach by means of which active packets on the fly deploy services as they carry executable code together with their data payload. This code will be dispatched and executed at designated nodes performing operations on the packet data as well as changing the current state of the node for packets that follow to find.
Both communities have reached now a mature stage that the first signs of convergence towards a single network programming architecture are evident. To this end, characteristics and methodologies developed by either of the two may be viewed as complementary rather than antagonistic each one making its own contribution to this new architecture.

2. PROGRAMMABLE AND ACTIVE NETWORKS

The motivation behind Opensig networks has been the observation that monolithic and complex control architectures are made up from basic components that can form a set of elementary network services. Modelling these services as individual objects by employing a high level programming language for the implementation and manipulation of the services-objects, results in a new generation of network architectures with some new desirable features: interoperability between different networking technologies, programmability for the creation of new services, and open interfaces across the control plane [A.3]
Eventually, a number of results out of the Opensig community were formalised by the IEEE Project 1520 standards initiative programmable network interfaces and its corresponding reference model [A.4]. P1520 advocates the need for open interfaces as the basis for service composition (creation). The interfaces are structured in a layered fashion offering their services to the layer above. Each layer defines what is termed as the level. Each level comprises a number of entities in the form of algorithms or objects representing logical or physical resources depending on the level s scope and functionality. This approach gives rise to the reference model (RM) depicted in the left part of Figure 1.

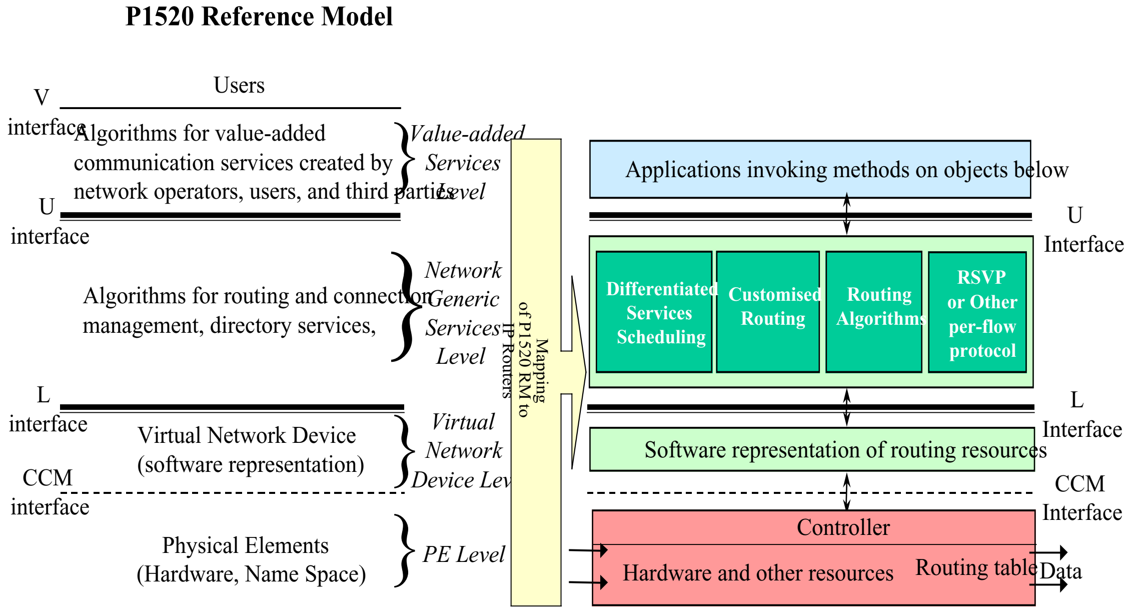
P1520 Reference Model
Users
V interface
Algorithms for value-added communication services created by network operators, users, and third parties
Value-added Services Level
U interface
Algorithms for routing and connection management, directory services,
Network Generic Services Level
L interface
Virtual Network Device (software representation)
Virtual Network Device Level
CCM interface
Physical Elements (Hardware, Name Space)
PE Level
Mapping of P1520 RM to IP Routers
Applications invoking methods on objects below
U Interface
Differentiated Services Scheduling
Customised Routing
Routing Algorithms
RSVP or Other per-flow protocol
L Interface
Software representation of routing resources
CCM Interface
Controller
Hardware and other resources
Routing table
Data

Figure 1: The P1520 Reference Model and mapping to IP routers

More specifically, P1520 RM distinguishes among the following four levels:

- The physical element (PE) level consisting of entities such as hardware and the device architecture that actually reflects upon the supported capabilities;
- The virtual network device level (VNDL) which logically represents resources in the form of objects (entities); isolating the upper layers from hardware dependencies or other proprietary interfaces;
- The network generic services level (NGSL) consists of entities in the form of distributed algorithms that bind (interconnect) together the objects of the VNDL level according to specific network functionality, e.g., routing, connection set-up;
- Finally, the value-added services level (VASL) includes entities in the form of end-to-end algorithms that enhance the generic services of the NGSL level; providing user-oriented features and capabilities in the applications.

The four levels give rise to four interfaces, namely, CCM (Connection Control and Management), L (lower), U (upper), V (value-add) interfaces. The CCM interface is actually a collection of protocols that enable the exchange of state and control information at a very low level between the device and an external agent. The L-interface defines an API that consists of methods for manipulating local network resources abstracted as objects. CCM and L-interfaces fall under the category of node interfaces. The U-interface mainly provides an API that deals with connection set-up issues. As in the case of the L-interface, the U-interface isolates the diversity of connection set-up requests from the actual algorithms that implement them. Finally, the V-interface (not shown in Figure 1) provides a rich set of APIs to write highly customised software often in the form of value-added services. Additionally, U- and V-interfaces constitute network-wide interfaces.

The P1520 Reference Model (RM) provides a general framework for mapping programming interfaces and operations of networks, over any given networking technology. Mapping diverse network technologies and their corresponding functionality to the P1520 RM is essential.
Initial efforts focused on telecommunication networks based on ATM and introduced programmability in the control plane, later they extended these principles to IP networks and routers. The right side of Figure 1 illustrates a mapping of the P1520 RM to IP routers. Nowadays there is an effort to create a framework for designing interfaces not just for routers but also for any network element like a router, or an optical switch etc that participates in forwarding traffic [A.5].

The innovation in Active Networks 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 from the job of processing packets is 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 [A.30], [A.17].
Separate mechanisms for loading and execution may be required for the control thereof. This separation enables network operators to dynamically download code to extend a node s facilities, 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 [A.26]. In this context, when a packet arrives in a node code and data are separated, code is loaded to operate on the packet s data or change the state of the node. A hybrid approach has been proposed [A.1].

Active Networks transform the store-and-forward network into store-compute-and forward. As in the case of P1520 active networks have also proposed their own reference architecture model [A.9] depicted in Figure 2.

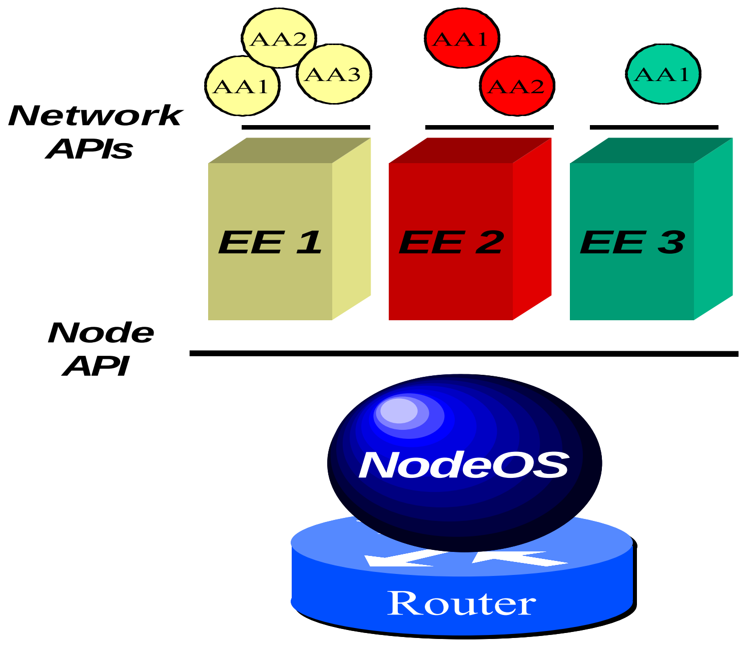

Figure 2: The Active Node Architecture

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 facilities are encapsulated by the Node interface through which EEs interact with the NodeOS. This is the minimal fixed point at which interoperability is achieved [A.2].

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 [A.10]. To this end, EE takes the form of a middleware toolkit for creating, composing and deploying services.

3. FAIN NETWORK NODES DESIGN

Conceptually an EE is the active network s programming environment [A.25] when instantiated it becomes the runtime environment of (or) a process [A.6]. This programming environment may be centred on a particular language and may export some API that encompasses elements like a Java Virtual Machine [A.25], [A.9]; toolkits used for building AAs (services) [A.30], [A.6] 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 [A.2]. The latter has an impact on where to draw the boundary between EE and NodeOS known as the Node interface.

In contrast, the AN reference architecture [A.9] is designed for simultaneously supporting a multiplicity of EEs at a node. This implies that EEs are treated as principals based on which authentication, authorisation and resource control takes places. Services and users that use an EE are hidden behind this principal allowed only to access NodeOS facilities through the EE they belong to. Prototypes proposed in [A.14], [A.12] fit in this picture.

Finally, EEs are 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 [A.7], [A.8]. One of the conclusions that we may draw from this analysis is that it is very difficult to come up with a common architecture that encompasses most, if not all, approaches as in the majority of cases the boundaries between architecture and implementation are blurred. Moreover, there is always a terminology abuse whereby the same term is used with different connotations.
On the other hand, such reference architecture is valid only if it provides explanations to most research efforts and is still valid to account for different implementation approaches. This is the subject of the next section.

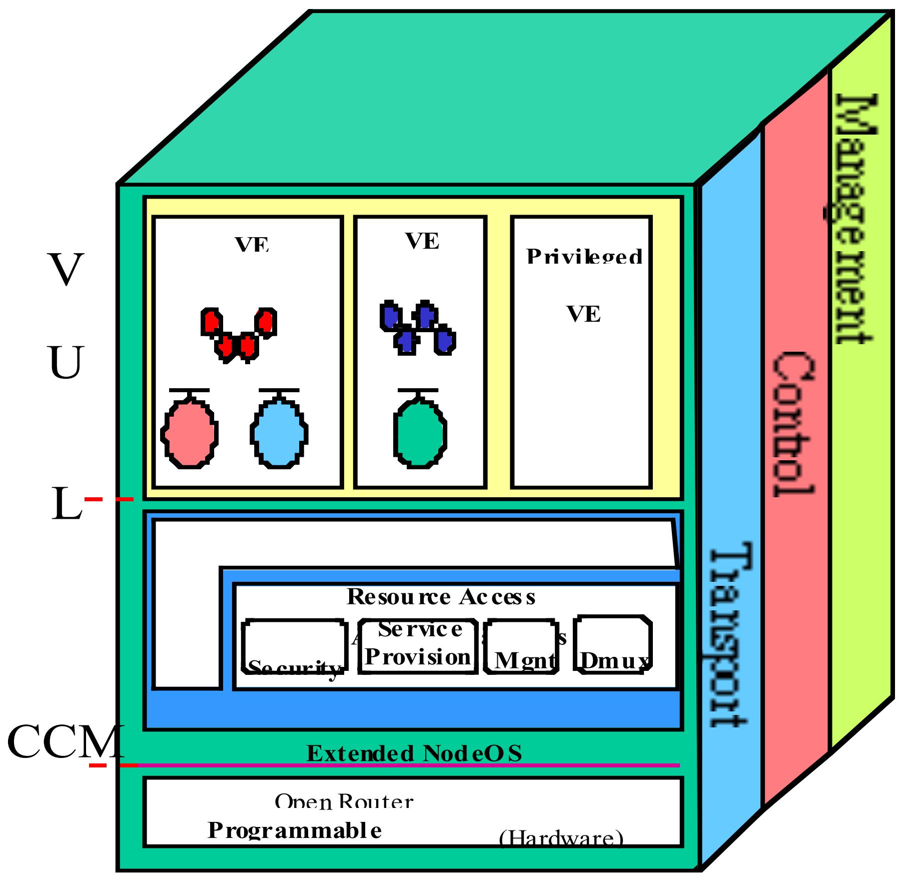

Figure 3: The FAIN reference architecture

In an attempt to establish a FAIN reference architecture independent of any implementation contexts we make the observation that EEs can be classified according to the following three categories: Technologies; Services; Virtual Environments

Technologies are all those toolkits like programming languages, specialised hardware, Java virtual machines, that allows someone to implement your designs in order to achieve the desired level of the requirements trade-off.

Services are what the EE has to offer usually in the form of static interfaces. By static interface we do not mean that the EE is also static. Services may well be extensible in the sense that EEs offer another interface (service) through which the service under consideration can be extended. For example a service uses the code distribution mechanism to download code extensions. The extension then becomes part of the service interface.

Finally, EEs act as virtual environments. This is an abstraction in itself as it is used for resource management and control. Therein services may be found and interact with each other. From the NodeOS viewpoint 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. From the viewpoint of customers and users all these three categories define the Network API exported by the EE [A.10].

Of course an EE may be treated as a mixture of all these categories but for the purposes of creating reference architecture we feel that this is confusing. Figure 3 depicts our proposal as the FAIN reference architecture. Any references to technologies and how these may be used to realise services or make the network programmable to achieve the desired level of programmability have been left out as we consider this an implementation issue.

Our approach is from the service viewpoint in the terms of interfaces and how these may be combined to compose other services thereby creating a set of building blocks to create a truly service platform. However, the methodology to compose services falls outside the scope of FAIN although this capability is necessary to increase re-usability and productivity. One way to go about is to adopt the P1520 approach and define U and V interfaces or use an active language like PLAN of Switchware. These services may solely lie in the control or management plane or may traverse all planes.

Services may be implemented anywhere in the node by any means of technology and they are invoked by their interfaces. To this end, architecture is separated from implementation that follows. In contrast, a virtual environment (VE) provides a place where services may be instantiated and used by a community of users and stay isolated from other communities. Within a VE many technologies may be used to implement and/or instantiate a service. However, using a diversity of technologies within a VE has a detrimental effect on interoperability requirement but this may be avoided by choosing a common interface language or representation that acts as a wrapper to the proprietary one. To this end, many technologies (active or not) may co-exist in the same VE and used in a synergistic way. The details of this synergy are an implementation issue but some preliminary ideas may be found in [A.2] when the concurrency model is discussed. When VEs are connected together they create a true virtual network. AN technology has advocated the co-existence of few Virtual Environments (EE) at each node where each one of them implements a different virtual machine on top of the NodeOS [A.30].

Another property of the reference architecture is that it makes no assumptions about how thin is a VE. 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 [A.14], [A.12]. Moreover, a VE may coincide with an implementation that is based only on one technology e.g. Java technology. In either case this is a design decision dictated by customer requirements. 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 are created. The network provider who has access rights to instantiate the requested VE on behalf of a customer should own this privileged VE. From this viewpoint 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 that are necessary to keep all the other components together, and support. The node OS can be viewed as a container of AN node services [A.21] called hereafter facilities. These facilities represent the local node services and provide the foundation for the execution of service components, which are usually in a network-wide scope. Example facilities are:

- 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). 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.
- Node External Interfaces / Network APIs components - Application Programming Interface (API) provides mechanism by which objects transparently make requests to and receive responses from other objects on different platforms in heterogeneous distributed environments like Active Networks. The API is object-oriented and consists of several categories of interfaces as follows: Service Interfaces offer applications access to a range of network capabilities; Framework Interfaces provide 'surround' capabilities necessary for the Service Interfaces to be open, secure, resilient and manageable; Administrative interfaces are to support administrative functions within the enterprise and to permit the supply of

services by third party vendors. A working version of the APIs will produced as IDL specifications.

To guarantee a secure and fair use of resources, the platform defines a resource control framework that partitions and allocates resources (including computing resources such as CPU time and memory, and network resources such as bandwidth and routing table). The framework implements the API as an abstraction of the partitioned resources, which will be used by an execution environment. It also implements a policing entity that enables policy-based management, i.e. enforcing and checking the access to node resources.

All these facilities in the NodeOS co-operate to deliver the overall functionality of the NodeOS to achieve its goals. One question that is relevant here is how much functionality should be put in the NodeOS and how much should be left to the VEs. This is related to the degree of extensibility a NodeOS is designed to achieve. For instance, a NodeOS may opt for a slim version whereby a sophisticated code deployment facility is not supported and leave this choice to the VE itself ending up with each VE using their own code deployment. The difference between the two approaches is that in the first case VEs do not have to start as empty boxes building everything from scratch while in the second case they can find facilities to build on top of them.

However, the exact choice has an effect on how the resource control is designed, as the NodeOS needs to charge the VEs for the use of these facilities since their use consumes resources. Note that limited extensibility in the NodeOS does not necessarily means limited extensibility at the Active node as it may realised in the VE possibly at the cost of the performance.

Between VEs and NodeOS lies the node interface that encapsulates all the facilities offered by it. Its objective is to create programmable abstractions of the underlying node resources, whereby third-party service providers, network administrators, network programmers or application developers can influence or extend node control through the use of higher-level API's. To guarantee a secure and fair use of resources, the platform defines a resource control framework that partitions and allocates resources (including computing resources such as CPU time and memory, and network resources such as bandwidth and routing table). The framework implements the API as an abstraction of the partitioned resources, which will be used by an execution environment. It also implements a policing entity that enables policy-based management, i.e. enforcing and checking the access to node resources.

The resource framework and the active network facilities will be designed as the services of a distributed processing environment (DPE). FAIN proposes to allow implementations of these services in different DPEs, depending on specific requirements in terms of performance, or functionality. DPE could be based on TINA-DPE [A.26], real-time ORB, JAVA virtual machine [A.31], mobile agent platform [A.23], or other distributed platforms. As a major contribution, an AN DPE will integrate a dedicated ORB and a mobile agent platform to provide a full range of signalling services and real-time QoS control for distributed applications. In order to support the needs of distributed multi-media and real-time bound applications, a specific execution environment will be provided which is optimised for high performance, i.e. high packet processing rates and low latency. At last, active network services can be provided to satisfy the very diverse requirements of applications in the future information society.

Node platform provides the basic functions on which execution environments rely. As such, it takes the form of an OS, manages the resources of the active node and mediates the demand for resources, including transmission, computing, and storage. It thus isolates VEs from the details of resource management and from the effects of the behaviour of other VEs. The VEs in turn hide most of the details of the platform from the users and implement the Network API.

The FAIN reference architecture is the starting post from which a detail node architecture specification will follow.

4. MANAGEMENT OF ACTIVE NETWORK

The use of policies for network management has recently been introduced to the Internet community. However, for the deployment of Policy Based Network Management (PBNM) Systems [B.9] in the Internet, a standardisation process is required to ensure the interoperability between equipment from different vendors and PBNM systems from different developers. Both the Internet Engineering Task Force (IETF) and the Distributed Management Task Force (DMTF) are currently working for the definition of standards for Policy Based Network Management. The DMTF is mainly focused on the representation of policies and the specification of a corresponding information model and schema. The IETF is also working in that field, in co-operation with the DMTF, while also trying to define a general framework for a PBNM system, as well as a protocol that could be used for implementing a PBNM system.

The routers today have had policy management filtering capabilities in them for years. Nonetheless, it has not been deployed on a larger scale until recently, when routers are enhanced to acquire Quality of Service (QoS) capabilities [B.2]. Policy management may soon become less of an option and more of a necessity because instituting QoS complicates network operations, as much as it involves mediation among various, and sometimes conflicting, network requirements. Additionally, as networks grow the amount of network elements that need to be managed grows. Not only are there more devices to be managed, but also the number of kinds of things (e.g., capabilities, services, types of interfaces, etc.) is growing [B.4]. As more kinds of network elements are introduced, so are more management interfaces the IT administrators must learn and use to manage the environment. In short, the challenge is to balance various claims on network resources, and to satisfy demands by users and network managers (and application developers) for higher levels of network security, predictability, and availability.

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

Active networks [A.29] can resolve many of the problems inherent in current PBNM technology. It allows the dynamically enhancement of the management architecture, the introduction of new application or device specific policies, tailored to realise complex tasks, and the automation of the network management tasks. The PBANEM architecture described in the following chapters shows how active networks could be used to overcome many management problems inherent in traditional management approaches. Customers ability to take part in the network management tasks is the first of such requirements and therefore the possibility to delegate the management responsibilities is, in fact, one of the operators expectations. Network operators would relish the promising potential that emerge with the advent of active networks in terms of rapid deployment of new services; customisation of existing service features; scalability and cost reduction in network and service management; independence of network equipment manufacturer; information network and service integration; and diversification of services and business opportunities.

5. FAIN NETWORK MANAGEMENT APPROACH

The management approach in the FAIN project is based on policies.
We envisage that the management of the active network will require the following ***components***:

- **Policies:** Design of management policies required to manage the active nodes and network

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

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

The Network and Service Providers will need to be provided with management capability at the active nodes to monitor and control all or part of the network.

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

- Element Management Nodes
- Network Management Nodes

The main difference in functionality provided by these two types of management nodes is in the policy types, which they could process and manage, in the sub-networks, which they cover and in the creation of management domains for different types of users, as shown the Figure 3.

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

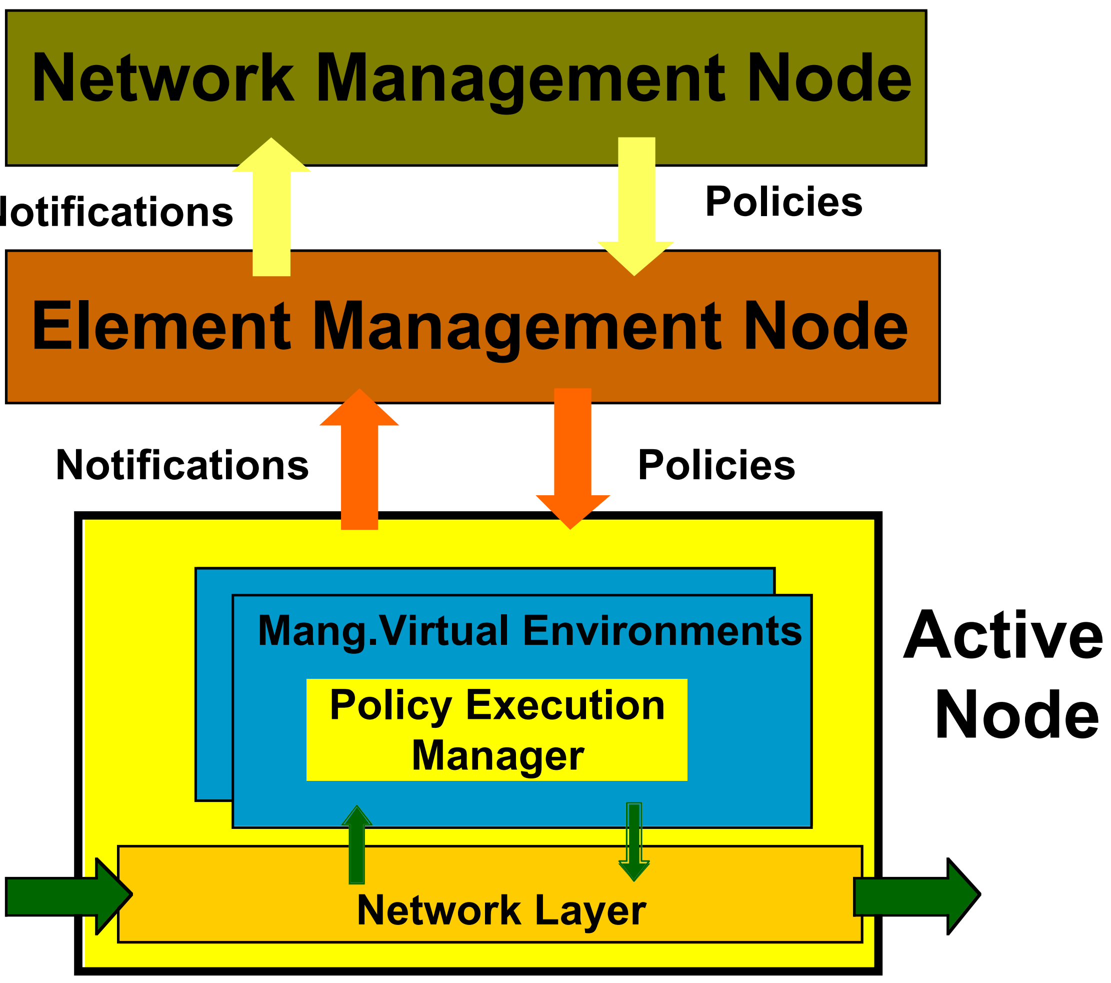

Figure 4 - Active Network Management

An active node provides an execution environment, which can run dynamically loaded software components. These software components, with characteristics allowing them to be loaded and run in a particular execution environment, are provided by the service / network operator of the active node and are therefore trusted code components. General users will not be permitted to load code onto an active router. The facility offered is the ability to set up software code via policies. User-defined policies will allow selection of software components from those provided by the service operator of the active node and configuration of certain parameters. This will facilitate the users to define their own packet labels for active processing at higher levels in the protocol stack.

All this management functionality will be developed at two levels, the network level, and the network element level. In FAIN there will be a small number, maybe just one, of network managers per domain that will implement the network level management functionality. While, the active network element manager will implement the network element level management functionality. We will have one element manager per active network element. The system described in this paper develops the network element level functionality of the Policy-based Active Network Management System developed in FAIN.

In a first classification we will distinguish to main fault types that should be considered when defining the management architecture: critical resource faults which limit the node operability leading to node collapses and therefore have a severe impact in network behaviour. This kind of faults endangers the local fault management itself. Secondly, non-critical resource faults, which do not imply a serious damage in node working, so that the local management system is able to proceed with the control functions and autonomously, recover from the failure. For example, this type of faults could lead to a performance fall. Several policy types could be defined in such a case to give priority to certain activities, normally those ones that are essential to avoid the node collapse. In both cases, it is necessary to implement the appropriate mechanisms for fault notification and local decision dissemination. In this way it is possible to modify the global policies that could be affected as a consequence of the local node anomalous behaviour.

Active networks could also be a means to enhance critical resource fault management since they provide the intelligence to foresee the possible faults and take rapid decisions as required. It could be possible even to dynamically and autonomously modify the policies to reassign resources or promote a smooth transition to another node. Traditional network management imposes inherent delays that prevent this kind of procedures. In the PDP-PEP interaction two main scenarios can be distinguished depending on the distribution of these components among the network nodes. In the first case, PDP and PEP appear in different network nodes whereas in the second one both components are integrated into the same node. Selecting any of the previous configurations will depend on the functionality that is to be provided, being both likely to appear in the FAIN project.

Given the active node complexity, it is foreseeable that its management may suppose the inclusion of several PEPs and PDPs. Therefore, it would be advisable to implement the protocol features which are unavoidable, exclusively for the provisioning and outsourcing operations, avoiding a major impact in node resources consumption and performance. The Policy Decision Point attends to two main aspects of the policy based network management. On one side, it takes charge of the retrieval of the policies coming from the Network Management Node or from the active applications, proceeding to their distribution to the appropriate PEPs. On the other side, it determines the policies to be applied in every moment depending on its knowledge of the node status. A set of components has been identified as essential for the PDP to accomplish these responsibilities. Local conflict detection and resolution is, in fact, one of the major functionality of the PDP. Including the Conflict detection block in the PDP is an essential condition since local conflict detection requires understanding the semantics of the policies (conditions and actions). Since several PDPs could coexist in the same active node, distribution of policies among the PDPs is a prerequisite to detect the conflicts that affect to each of them. Since not every policy has to be distributed to each PDP, different PDPs could have been provided with different sets of policies.

In order to tackle with criteria not even known at PDP design time (e.g. those coming from active applications) we pretend to use an active approach. The active packets containing the metapolicies (data) will also contain the condition or action interpreters (code) required to execute the policy, which are not in the system yet. These interpreters will be treated as the pieces or building blocks of the policy interpreter. This way, the PDP can acquire knowledge about the policy classes used by the PEP. Using this scheme, the same PDP type can be used for managing different types of enforcement clients.
The computational resources of the node are the processing power (CPU cycles), the memory and the disk storage. An active node may have multiple Execution Environments. These EEs are competing for the resources of the node and therefore the administrator must set up the corresponding policies to partition the computational resources of the node as needed. There is also a need to assign packet flows to EEs. The PBNM system is also responsible for the creation, deletion and modification of EEs. When the creation of a new EE is requested, the PBNM system should check existing policies, to see if the person that made the request has the necessary privileges and if there are sufficient resources in the node for this EE. There should also be priorities for different EEs, to ensure that the most critical EEs always have enough resources to execute. These priorities also depend on Service Level Agreements made between the Network Provider and the Consumers.

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

A. Configuration Management

The configuration of the system can be done in three ways: provisioning, signalling and self-adaptation.
In the provisioning scenario, the network administrator defines policies using a tool application and sends them to the PBNM System. The Security Checks component first examines if an authorised user has defined the policies and also checks the syntax and any possible conflicts with other policies. The PEP Manager gets the policies, after they have been validated by the Security Checks component. Before registering the policy conditions and downloading the necessary actions to the PEPs, the PEP manager should check if the target

nodes of the policy have the adequate functionality to support its actions and conditions. This means that the PEP Manager should maintain information about the capabilities of the target nodes.

If a node does not have the capabilities to enforce a given policy, then an error message should be produced. If the node can support the policy, then the PEP manager should also check if the existing PEPs have the required functionality. If not, a new PEP should be installed on the node. The necessary information model is defined by the IETF, although it may require extensions to allow the configuration of AN-specific resources, like resource reservation for EEs, scheduling algorithms, assign priorities to certain EEs etc. The resources which are going to be manageable also depend on the FAIN Resource Control Framework, which should provide an interface to manage and configure AN resources.

From the configuration management point of view the main tasks that the PBANEM system should perform are as follows:

Creation, deletion and modification of paths with a certain quality of service: the network element management system should be able to create, delete and modify paths through the network element that assure a certain quality of service for a flow. To be able to develop this task the management system should be able to: identify packet flows according to packet fields; reserve (and free) bandwidth resources to flows; and access and modify the routing table on a per-flow granularity.

Creation, deletion and modification of execution environments: customers might ask the management system the creation of an execution environment in a number of nodes. The customers will run code within their execution environments that will control or process their packets in a certain way. The execution environments should have assigned a certain amount of computing resources in order to process this code. The tasks of create, delete and modify an execution environment within the active node raises the necessity of the element management system to access certain resources of the active node. It should be able to:

Assign computing resources (CPU cycles, memory.) to the execution environment: this will involve the modification of the resource control framework that controls the access of EEs to node resources.

Assign flows to EEs: the customer code within the EE might want to process its packets in a certain way

Install and remove code within the EE: The Customer might ask the management system to install a certain code within their EEs

The reservation request for both computing resources and forwarding resources can come from the management system or from the active node (signalling protocol). In order to support this second alternative the active node interface should be able to support the ability of sending reservation request to the PBANEM system.

B. Fault Management

The fault management functionality that will be developed by the PBANEM system is simply that of reporting to the network management level the unexpected alarm events that might occur in the active node. The management system might decide to enforce some recovery actions in the affected active nodes. Therefore, the unique requirement imposed on the active node interface offered to the management system is the ability of sending asynchronous alarm messages when an unexpected alarm situation occurs. Some of these alarm situations might be congestion, lost of signal in a link, end of memory available in the node, CPU saturation, etc. Optionally, might be interesting that the interface offers the possibility of choosing which situations will cause an alarm to be sent. However, this is not a mandatory requirement.

C. Performance Management

The performance management functionality developed by the management system is that of monitoring a number of active node characteristics or resources in order to obtain statistical information. Therefore, the interface of the active node should offer information of the status of this resources or characteristics. Obviously, as much information is given better statistical information can be recompiled. However the most important information that should be available is information about node computing resources status like CPU use percentage, use memory, free memory, and available hard disk. Moreover, this information should be given in general and per flow, number of threads in the node, number of threads per execution environment, etc.

The PBANEM system decide when policies should be applied base on the following information: node forwarding resources status like bandwidth use percentage per interface, free bandwidth per interface, used bandwidth per interface, used bandwidth per flow, dropped packets per interface, dropped packets per flow, and queue status

D. Accounting Management

The information obtained from the monitoring of node resources listed in the previous section, as well as the Configuration Management information that the management system has for each component will be sufficient for realising the accounting management tasks within the PBANEM system.

E. Security Management

From the security management point of view, we have to consider two aspects. Firstly, the security of the active node itself is in question. In order to avoid that a non-authorised person manipulates node resources, the interface should only be accessible from the exterior for the management system. Another important security requirement is that the interface should be able to provide authentication information of the requesting principal in whose behalf a reservation request is made from the active node to the management system. This authentication information will avoid the reservation of resources to a principal who does not have the appropriate reservation rights.

6. Policy Implementation

Authorisation Policies: They define the actions that a network manager, say, can perform on the objects in the target domain [B.14]. It can be described as an access control policy to protect resources and services from unauthorised access, and are implemented on the target host by an access control component.

Information Filtering Policies: This category is needed to transform the information input and output parameters in an action. Although it may seem like an authorisation policy, it differs from the normal ones in that it is not possible for an external authorisation agent to make an access control decision based on whether or not an operation, specified at the interface to the target object, is permitted.

Delegation Policies: Delegation is often used in access control systems to cater for the temporary transfer of access rights. However, the ability of a user to delegate access rights to another must be tightly controlled by security policies. This requirement is critical in systems allowing cascaded delegation of access rights. This policy specifies the authority to delegate; it does not control the actual delegation and revocation of access rights.

Refrain Policies: These policies define the actions that subjects must not do on target objects even though they may be actually be permitted to perform the particular action. They are used for situations where negative authorisation policies are in appropriate, i.e., when the subjects do not trust the targets to enforce certain policies.

Obligation Policies: Obligation policies specify the actions that must be performed by managers within the system when certain events occur and provide the ability to respond to changing circumstances. They are event-triggered, and define the actions subjects must perform on the target domain.

7. CONCLUSION

This paper reviews the characteristics and methodologies developed by the programmable network and active networks communities as complementary approaches making a significant contribution to the development of new architectures. It reviews a policy-based network management approach to the management of active networks. One such active network and management architecture is developed in the IST Project FAIN [A.19], [A.20]. The FAIN policy based network management is described. FAIN is a three-year project, whose main task is to develop and validate an open, flexible, programmable and dependable network architecture based on novel active node concepts. The generic architecture for active networks is an innovative integration of active networking, distributed object and mobile agent technology.

8. ACKNOWLEDGEMENTS

This paper describes work undertaken and in progress in the context of the FAIN — IST 10561, a 3 year project during 2000-2002. The IST programme is partially funded by the Commission of the European Union. The FAIN consortium consists of University College London (UK), Jozef Stefan Institute (Slovenia), NTUA (Greece), Universitat Politecnica de Catalunya (Spain), Deutsche Telekom Berkom (Germany), France Telecom (France), KPN (The Netherlands), Hitachi Europe Ltd. (UK), Hitachi Ltd. (Japan), Siemens AG (Germany), ETH (Switzerland), GMD Forschungszentrum Informationstechnik GmbH (Germany), IKV++ GmbH (Germany), INTEGRAsys (Spain), University of Pennsylvania (USA).

9. References

A. *Active and Programmable Networks References*

[A.1] Alexander, D. S., W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith, "The Switchware Active Network Architecture", IEEE Network Special Issue on Active and Controllable Networks, vol. 12 no. 3, May/June 1998.

[A.2] AN Node OS Working Group, "NodeOS Interface Specification", January 2000

[A.3] A.A. Lazar, K-S. Lim, and F. Marconcini, "Binding Model: Motivation and Description", 1995

[A.4] Biswas, J., et al., "The IEEE P1520 Standards Initiative for Programmable Network Interfaces", IEEE Communications, Vol. 36, No 10, October 1998. http://www.ieee-pin.org/

[A.5] Biswas et al., "Proposal for IP L-interface Architecture", IEEE P1520.3, P1520/TS/IP013, 2000

[A.6] Berson, S., et al. "Introduction to the Abone", 2000

[A.7] Bhattacharjee, S., "Active networks: Architectures, Composition, and Applications", Ph.D. Thesis, Georgia Tech, July 1999

[A.8] Braden, B., A. Cerpa, T. Faber, B. Lindell, G. Phillips, and J. Kann, "ASP EE: An Active Execution Environment for Network Control Protocols", ISI Technical Report, December 1999.

[A.9] Calvert, K. L., ed. "Architectural Framework for Active Networks", Version 1.0, Active Network Working Group, July 1999

[A.10] Calvert K. L., S. Bhattacharjee, E. Zegura, and J. Sterbenz, "Directions in Active Networks", IEEE Communications Magazine, October 1998.

[A.11] 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

[A.12] Campbell, A. T., H. De Meer, M. E. Kounavis, K. Miki, J. Vicente, and D. Villela, "The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures",2nd IEEE International Conference on Open Architectures and Network Programming (OPENARCH'99), New York March 1999

[A.13] S. Denazis, K. Miki, J. Vicente, A. Campbell — "Designing Interfaces for Open Programmable Routers" in the Proceedings IWAN'99 - Berlin June 1999, Springer ISBN 3-540-66238-3

[A.14] Van der Merwe, J.E., S. Rooney, I.M. Leslie, and S.A. Crosby, "The Tempest: A Practical Framework for Network Programmability", IEEE Network, Vol. 12, No. 3, pp. 20-28, May/June 1998

[A.15] CORBA www.corba.org

[A.16] DARPA Active Networks Programme — www.ito.darpa.mil/researck/anets

[A.17] Decasper, D., G. Parulkar, S. Choi, J. DeHart, T. Wolf, B. Plattner, "A Scalable, High Performance Active Network Node", IEEE Network, January/February 1999

[A.18] Hicks, M., et al., "Experiences with Capsule-based Active Networking", Tech. Report, University of Pennsylvania, 2000

[A.19] FAIN project WWW Server — www.ist-fain.org

[A.20] A. Galis, B. Plattner, J. M. Smith, S. Denazis, H. Guo, C. Klein, J. Serrat, J. Laarhuis, G.T. Karetsos, C. Todd "A Flexible IP Active Networks Architecture" in the Proceedings Second International Working Conference, IWAN'2000 — Japan, October 2000, ISBN 3-540-41179-8, Springer Verlag

[A.21] IETF "An architecture for Differentiated Services" S. Blake, August 1998 http://search.ietf.org/internet-drafts/draft-ietf-diffserv-arch-01.txt

[A.22] ITU-T Recommendation Q.1201 "Principles of intelligent network architecture"-1992; Recommendation O.1224 "Distributed functional plane for intelligent networks-CS-2" —1997; Recommendation Q.1225 "Physical plane for intelligent network CS-2"-1997; Recommendation Q.1211 "introduction to intelligent network CS-I"- 1993; Recommendation Q.1229 "Intelligent network user s guide for capability set 2"- 1997

[A.23] Multiservice Switching Forum (MSF). http://www.msforum.org/

[A.24] Open Signalling Working Group, http://www.comet.columbia.edu/opensig/.

[A.25] RFC 2475, "An Architecture for Differentiated Services", 1998

[A.26] Smith, J. M., et al., "Activating Networks: A Progress Report", IEEE Computer, 1999.

[A.27] Schwartz, B., A. W. Jackson, W. T. Strayer, W. Zhou, D. Rockwell, and C. Partridge, "Smart Packets for Active Networks", OPENARCH'99, March 1999

[A.28] TINA www.tinac.com

[A.29] D. Tennenhouse, D. Wetherall — "Towards an active network architecture" Computer Communications Review, 26, 2 (1996), pp 5-18

[A.30] D. J. Wetherall, J. Gutter, D. Tennenhouse — "ANTS A toolkit for building and dynamically deploying network protocol" IEEE OPENARCH, April 1998.

[A.31] World Wide Web Consortium, Extensible

B. Network Management for Active and Programmable networks

[B.1] M. Brunner, R. Stadler, Virtual Active Networks — Safe and Flexible Environments for Customer-managed Services , Tenth IFIP/IEEE Interna tional Workshop on Distributed Systems: Operations and Management (DSOM 99), Zurich, Switzerland, 1999.

[B.2] T. Rybczynski, Policy-Enabled Networking: What's It All About? , CTI Inside Networking, January 1999

[B.3] H. Mahon, Yoram Bernet, Shai Herzog, John Schnizlein, Requirements for a Policy Management System , Internet Draft, November 9, 2000

[B.4] M. Brunner, B. Plattner, Management of Active Networks , ICC Workshop on Active Networking and Programmable Networks, Atlanta, 1998

[B.5] M. Stevens et. al., Policy Framework , Internet Draft, March 2000

[B.6] E. Lupu, A Role-Based Framework for Distributed System Management , Departement of Computing, Imperial College, July 1998

[B.7] K. Ho Chan, D. Durham, S. Gai, K. McCloghrie, F. Reichmeyer, J. Seligson, A. Smith, R. Yavatkar, COPS Usage for Policy Provisioning , Internet Draft, October 2000

[B.8] Morris Sloman, Jorge Lobo, Emil Lupu — Policy for Distributed Systems and Networks — Proceedings to Policy2001 Workshop, Bristol, January 2001- Springer Verlag, ISBN 3-540-41610-2

[B. 9] CORBA www.corba.org

[B.10] A. Galis, D. Griffin, W. Eaves, G. Pavlou, S. Covaci, R. Broos — "Mobile Intelligent Agents in Active Virtual Pipes" — in "Intelligence in Services and Networks" — Springer Verlag Berlin Heildelberg, April 2000, ISBN 1-58603 007-8

[B.11] TINA www.tinac.com

[B.12] World Wide Web Consortium, Extensible Markup Language www.w3.org/XML

[B.13] N. Damianou, N. Dulay, E. Lupu, M. Sloman, The Ponder Policy Specification Language