The Octopus Network Model: Opening Up the Internet to Active and Programmable Network Implementations

Carlos Macián

University of Stuttgart
Institute of Communication Networks and Computer Engineering
Pfaffenwaldring 47, 70569 Stuttgart, Germany
macian@ind.uni-stuttgart.de

Abstract. The inclusion of innovative services in commercial networks is a burdensome task which frequently encounters resistance from Network Operators. Opening up the network is a prerequisite for the Active & Programmable Network paradigm to succeed. In this paper we present a novel network model which addresses three critical points to achieve that goal: network security and safety, service management and high performance. We show that excessive virtualization of network resources penalizes performance and we introduce programmable hardware at the core of our model. We also introduce a two-tier security checking architecture which frees network nodes from the most heavyweight tasks, improving performance. Our single point of service admission permits strict security control. Lastly, the separation between service introduction and service management increases network flexibility and permits the smooth integration of other network architectures in our framework. We also present the Octopus Open Gateway architecture, which shall support our network model.

1 Introduction

Although networking is a highly dynamic field, a broad consensus exists regarding the difficulties of transporting innovative concepts into real networks. The deployment of new services or the introduction of new protocols is a slow and burdensome task, in which the Network Operator can be seen as the bottleneck. The reasons are manifold: On the one hand, his traditional sources of revenue, the transport of data and the management of the network, are becoming a commodity. Increasing competition is driving prices down while the requirements posed by clients are always increasing. More functionality, more bandwidth and better service imply costly investments to keep up with the latest technology, which quickly becomes obsolete. On the other hand, the heterogeneity and complexity of modern networks makes its management and configuration increasingly complex. It is very difficult to foresee the implications of introducing a new service or protocol for the correct behavior of the network as a whole before testing it on the field. As a consequence, the Network Operator is discouraged (for economic as well as technical reasons) from expanding the functionality of its network and especially from granting third-parties

I.W. Marshall, S. Nettles, and N. Wakamiya (Eds.): IWAN 2001, LNCS 2207, pp. 152-163, 2001.

access to its management. Its main concerns to "open up" its infrastructure are thus the security, safety and performance of the network.

The Active and Programmable Network (A&PN) communities, on their side, defend the idea that to foster innovation two requisites are necessary: Allowing Service Providers direct access to, and (partial) control of, the nodes and making network nodes programmable. The Active Network community goes even further by introducing the packet as the main network control and configuration unit: Incoming packets shall trigger the activation, download or reconfiguration of services inside the nodes. Several proposals in these fields have shown the feasibility of the concepts as well as some of their advantages. Nevertheless, it is our claim that for the broad dissemination of the A&PN paradigm three problems remain unresolved: A satisfactory security model, a general service management model and high performance.

Existing proposals either do not provide a general framework addressing the security concerns of Network Operators or do so by developing heavy security architectures that strongly penalize performance. Although there is some work in progress trying to surmount this conflict, we believe that no existing architecture has achieved it so far in a completely general way.

The second unresolved problem is performance. On the one hand, sharing control and communication network resources among several parties, as A&PN defends, needs coordination in the form of middleware actors, resource managers and the like. All this additional elements have a negative effect on performance. On the other hand, as already mentioned to fulfill the security requirements of this open networks burdensome procedures are needed. We nevertheless claim that it is the exclusion of open hardware what will most negatively affect performance in the long run. By emphasizing an abstract view of network infrastructure, present approaches prevent service developers from taking direct advantage of the node hardware. There are many applications that would profit from hardware support. The increasing speed of the networks and the tremendous development of programmable hardware show us the necessity and the feasibility of using hardware-software co-design of new services to successfully support network programmability.

Lastly, open network programming interfaces (ONPIs) of some sort are at the core of most proposals. It is claimed that they provide a foundation for service programming and the introduction of new network architectures. Beyond this undeniable fact, the problem of evolving ONPIs remains. Since it is impossible to foresee all the ways in which networking might evolve, programming interfaces, if not very carefully designed, are in themselves a restriction to innovation. They constrain the ways in which service creation and management might develop.

In this paper we introduce the Octopus Open Network Model, which we believe addresses the three points stated above. We have developed a node architecture that includes not only a programmable software environment for Service Providers, but also a programmable hardware platform. This should strongly improve performance. We address the problem of the evolution of network programming interfaces by clearly differentiating service introduction from service management. We moreover keep the standardization of the latter at a minimum. We also use the introduction of our Trusted Development Servers (TDSs) to structure our security architecture in two tiers, placing all heavyweight mechanisms at the TDS. This should also benefit performance at the nodes.

The rest of the paper is structured as follows: In chapter 2 we summarize some of the most relevant previous work. In chapter 3 our network model is presented, while in chapter 4 our node architecture is described. We conclude the paper in chapter 5 by summarizing our contributions and presenting some future topics of research.

2 Previous Work

Research in the areas of Active and Programmable Networks has already produced a broad set of proposals. We will not exhaustively examine them here, but we will concentrate on the most relevant ones for our work instead. For a good survey on this area we refer the reader to [3] and [9].

The inclusion of programmable hardware in active node architectures has been rare. The idea is nevertheless present, as has been shown in the work of Hadzic et al. at the University of Pennsylvania [7] and Decasper et al. at Washington University at St. Louis [4] and recently also in the work of Dr. Zitterbart´s group at the University of Braunschweig [8]. The P4 architecture, developed under the Protocol Boosters project [6] at UPenn consists of a pipeline of FPGAs interconnected by a switching array and controlled by a special Controller Unit. The Controller decides to which FPGA an incoming packet should be sent and also permits on-the-fly reprogrammability by dynamically separating any FPGA from the pipeline. With their prototype, Hadzic and his colleagues showed the feasibility of the idea, although in a very restricted form, since their platform was not designed to support several services concurrently or to dynamically select which packets should be processed by a certain service.

The FHiPPs platform developed at the University of Braunschweig presents a much more advanced structure. It also includes several FPGAs interconnected by a switching matrix, plus an external processor, a DSP and ATM interfaces. The hardware is accessible from any application via so-called Happlets, which extend the functionality of classical device drivers to manage the reconfigurability of the platform. In itself a very promising design, it is nevertheless not clear how different packet streams shall request processing by different services multiplexed onto the same platform or how a chosen packet stream should access several services in a row. Moreover, the design is in itself monolithic, without expansion possibilities.

The ANN design from WashU is the most comprehensive of all, including all aspects of the node architecture. Their particular hardware is composed of a set of ANPEs (Active Network Processing Engines) interconnected by an ATM switch core. Every ANPE includes a CPU, a FPGA and some memory. These elements are controlled by the node OS, which can reprogram any of them on-the-fly. Scalability is provided by means of attaching more ANPEs to the switch. The only limitation is in the surveillance of the shared resources in hardware. As we will discuss later, software control of hardware resources is not enough in presence of malfunctioning or greedy service designs.

On the software side, we borrow heavily from the experience gained by the groups at UPenn and WashU, plus the Tempest framework at the University of Cambridge [10]. The Switchware project at UPenn [1], [2] attempts to balance the flexibility of programmable networks and the security requirements stated in previous chapters.

The main elements of their architecture are active packets, dynamically loadable programs called Switchlets and active nodes. Active packets are written in a safe language called PLAN. In order to ensure safety the actions that active packets can realise are very restricted. When more complex tasks are needed, active packets can call Switchlets, which are programmed in CAML. This language supports formal methodologies to prove security properties of the code. This code segments are loaded out-of-band into the node. At the lowest layer, the Secure Active Network Environment (SANE) ensures the integrity of the entire environment.

The DAN architecture at WashU [5] sees services as a set of functions that are called by incoming packets. A packet might call several functions, which are then daisy-chained to process the packet in a row. If a packet needs a function which is not present in the node at the moment, it is downloaded from a well-known code server. This introduces additional delay, but permits to concentrate the most heavyweight security checks in those servers, where new modules are first stored. The server authenticates itself when downloading a new program into a node. The module itself can also be digitally signed. We elaborate on the idea of code servers to develop our Trusted Development Servers (see chapter 3).

The most characteristic item of the Tempest framework is the definition of several parallel control architectures over the same infrastructure. This control architectures are furthermore customizable on a per-service basis with the help of mobile code. The whole concept rests upon the abstraction of node resources, which permits to share them in a transparent way among coexisting control architectures, under the common surveillance of a resource divider called Prospero. This view of virtual networks over the physical infrastructure are at the core of our Logical Overlay Networks (LONs). The Tempest is nevertheless restricted to ATM networks and we will contend that their degree of resource abstraction penalizes performance.

3 The Octopus Open Network Model

The A&PN paradigm implies that the Service Provider is going to become the principal actor in the networking world. He will provide the content to give added value and differentiation to any network. Furthermore, it is the Service Provider itself who is going to manage its services. The Network Operator will find itself reduced to a commodity provider, which in this case means providing connectivity, bandwidth and a set of general management services and surveillance of the network. Certain basic QoS guarantees also fall into this category.

This new network model, then, foresees the Service Provider as the Operator of its own Logical Overlay Network (LON), formed by all the (node and network) resources used by its services. Many such Service Operators will then be multiplexed over the same physical infrastructure. This model implies a new set of relationships among networking actors[1] (see Fig. 1).

[1] Although regulators (governmental agencies, standardization bodies, etc.) certainly influence all actors, their role will not be further explored here.

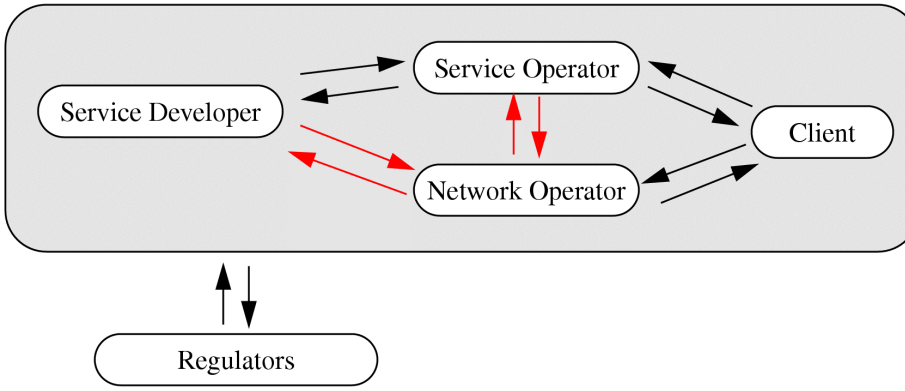

Fig. 1. The Relationship among Actors.

The client will be dependent on the Network Operator for his connectivity and basic transport of data and on the Service Provider for the content. At the other extreme of the chain, the Service Developer depends on the capabilities of the network, which fall under the control of the Network Provider, to develop new services. But he also must adapt his design to the necessities and business model of the Service Provider.

Nowadays the control and management of the network still lies in the hands of the Network Operator. In the end, it is the way and the extent in which the Network Operator will open its network what is going to set the transition speed to, and the ultimate success of, this new environment. We claim, as stated above, that innovation is being slowed by the rigidity of the networks. In order to accelerate this transition, a model is needed that guarantees the Network Operator the ultimate control of its network while letting Service Operators freedom to innovate.

The main elements of the Octopus Open Network (OON) Model can be seen in Fig. 2.

First among those is the Octopus Open Gateway (OOG), which will be analysed in chapter 4. These nodes are shared among many Service Operators. The union of all resources used by any one Service Operator is called a Logical Overlay Network (LON)[2]. There are two access points to a LON. The first one is the Trusted Development Server (TDS). It represents the interface to introduce new services into a LON. The interface to manage those services, once installed, is directly controlled by the Service Operator (dotted line on Fig. 2).

[2] We avoid the common terms “virtual node” and “virtual network” because we find them flawed. As so often in the networking world, we consider here simply an abstraction, a logical view of a physical infrastructure. There is nothing virtual about it.

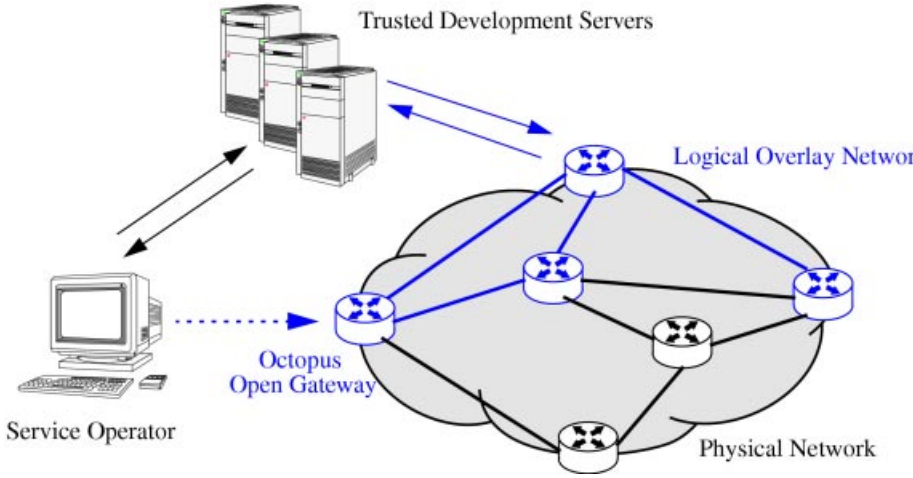

Fig. 2. The Octopus Open Network Model. Main Elements.

The OOG provides a software and hardware programmable platform. It is our goal to foster the quick development and installation of new services by improving portability and taking the Network Operator off the management path. We also foster performance by opening the node hardware to the Service Developer. We acknowledge, though, that network heterogeneity is here to stay. Hence, complete portability, especially for hardware modules is impossible to achieve. We shall elaborate on this shortly.

The development and insertion of a new service is as follows:

First, the Service Developer (possibly under contract of a Service Operator) designs a new service. We foresee the use primarily of platform-independent languages, for the software parts of the service (with Java as an example) as well as for the hardware parts (e.g. VHDL). Since the architecture of every node is different and does not fall under control of the developer, only the most abstract description of the hardware modules of a service can be kept portable. In a second step the developer must adapt his hardware modules to the concrete platforms where it is going to run.

To introduce the service in the LON, the code is then sent to the TDS. We foresee the deployment of at least one TDS per Equipment Provider and Network Operator. The role of the TDS is to check the rightness of the code by formal methods and to apply the most heavyweight security checks on the new service and its provider. The second role of the TDS is the integration of the new service in the configuration of the nodes. This mainly consists in communicating to the node the resource requirements of the new service, in order to check if they can be satisfied. This resources are mainly CPU time, memory space and bandwidth. Since a service can be formed by software and hardware modules, the task of resource monitoring at the node is performed by the NodeOS as well as by our Hardware Manager, described in chapter 4. Since the Hardware Manager is directly implemented in form of VHDL code in our FPGAs, its reconfiguration is in fact performed at the TDS, which integrates the resource requirements in the new configuration of our hardware platform. For that,

the TDS must either have a local copy of the configuration of its associated nodes, or upload it as needed. It would be advisable to have a hierarchy of TDSs in the networks, in order to more efficiently distribute the load and to prevent a single point of failure.

The TDS realizes the adaptation of new services to the nodes. A TDS is in charge of this for every Service Operator using a certain physical infrastructure. Thus, strong security measures inside the TDS are needed in order essentially to prevent access to foreign code. We envision the deployment of secure OSs to accomplish that.

The adapted service is then downloaded into the relevant nodes. In case that a LON consists of nodes of different Equipment Providers, this adaptation must be done in parallel at their respective TDSs.

As we see, the role of the TDS is twofold: First, it implements the secure interface to the introduction of new services. Doing that, it takes the burden of most security checkings off the node, thus improving performance. The node must simply authenticate TDS and code prior to accepting a download. Its second task is adapting the code to its environment and communicating the resource needs of the new service to the node. This precludes the need for costly (in terms of money and performance) additional processing power inside the nodes.

Once a new service has been deployed, its management falls entirely in the hands of the Service Operator. The resources that the service is going to use have already been set at the TDS and are hard-coded inside the NodeOS and Hardware Manager. This two entities monitor the behavior of all services installed in a node, enforcing their correct behavior. Hence, there is no need to restrict the ways in which the Service Operator manages its service. No restrictive interface is needed. This characteristic of our model can be seen by some as a drawback, since it provides the finest possible granularity of service control. Some Service Operators might prefer to have a certain management support. Our model does not preclude specialized companies from providing those services. For those other Service Operators which prefer to keep absolute control over their services, no restrictions are imposed.

We thus avoid the standardization of Network Programming Interfaces (NPIs), since we believe that no NPI can foresee all possible technological evolutions. Hence, any NPI represents a potential restriction to innovation. That is why we substitute this concept by our Service Admission Interface (SAI), implemented in our TDS.

4 The Octopus Open Gateway Architecture

The node architecture that we have developed is presented in Fig. 3.

It consists of three main blocks: A management CPU, a Basic Hardware Platform (BHP) and a Universal Hardware Platform (UHP).

The BHP implements the basic communication functionality, i.e. it is a “plain” router. Those incoming packets that do not need any kind of special treatment will simply be forwarded in a traditional fashion by the BHP. Nevertheless, at different points of the processing path (after packet classification, route lookup, etc.) there exists the possibility to forward the packets to the UHP via an internal backplane. The UHP presents the programmable hardware platform that Service Developers can use

for their designs. The separation between BHP and UHP guarantees backward compatibility, since we do not force any change in the format or function of packets.

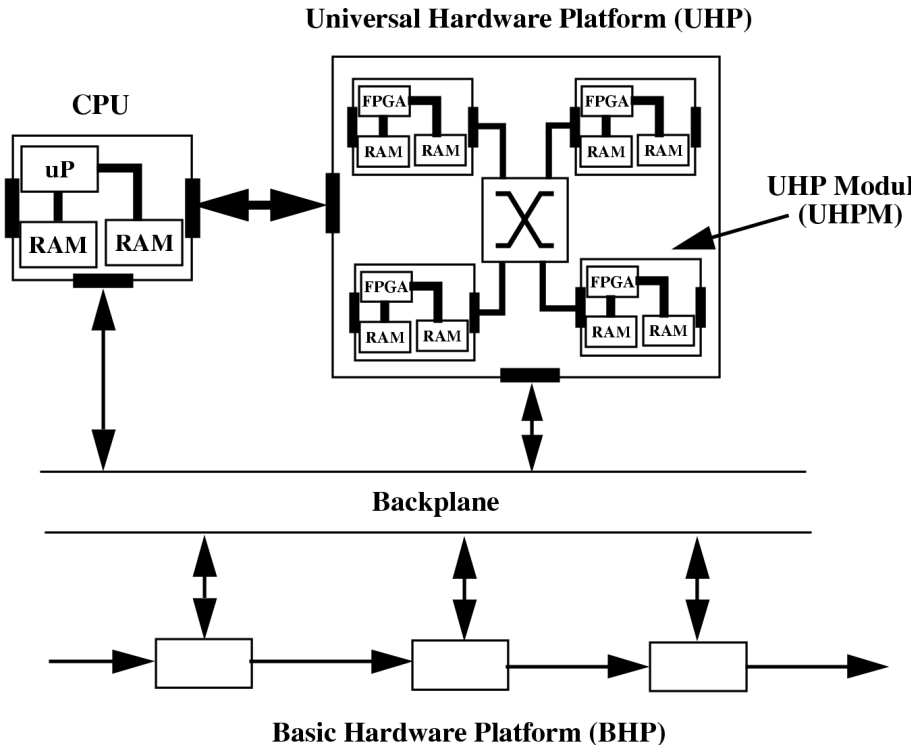

Fig. 3. The OOG Hardware Architecture.

The UHP is composed of several modules, so-called UHP Modules (UHPM) interconnected by another backplane. Every UHPM initially contains an FPGA and some memory. No further interfaces are needed, since the communication with the CPU and the BHP is controlled by a special unit in the UHP. The scalability of our platform is guaranteed by the modularity of the design. More UHPMs can be added at any time. On the other hand, since we specifically separate BHP and UHP, the later can be substituted for a more powerful one if needed.

The CPU supervises the functioning of the node and provides the software environment where new services will be inserted. Lately, the possibility to integrate microprocessor cores directly in the FPGAs has arisen. This seems very attractive, for it allows the integration of both parts of a service (software and hardware) in a common platform, easing the interchange of information between them. We leave this option for further study.

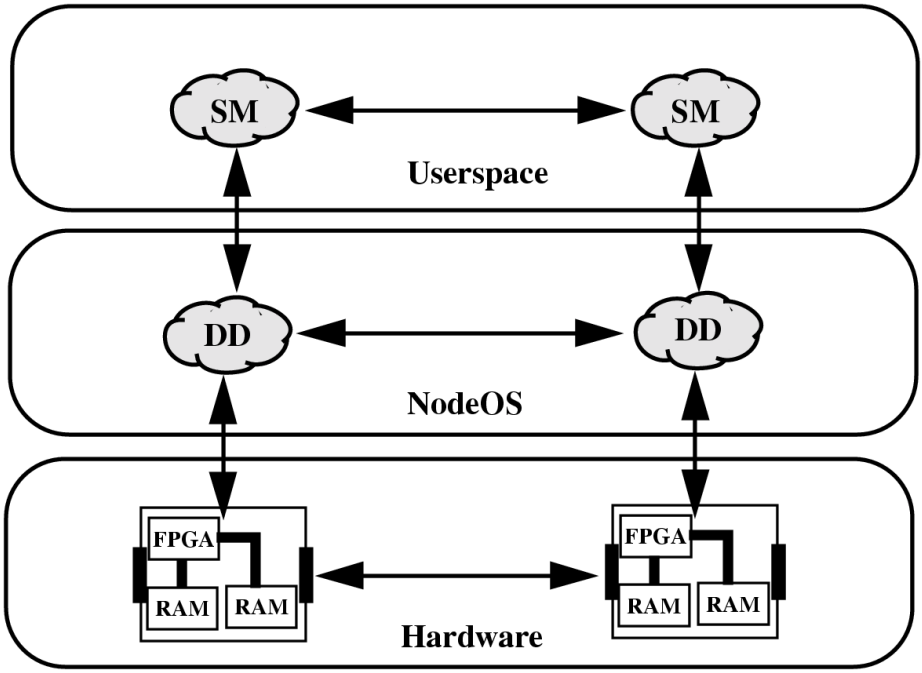

Fig. 4. The Service Structure.

The structure of a service can be seen in Fig. 4. As already mentioned, a service can be composed of several software modules (SMs, a.k.a. applications) and several hardware modules, which directly run on the UHPMs. They communicate by means of specialized device drivers (DDs). In this area we are investigating the possibilities of the Happlets introduced in the FHiPPs project [8]. To guarantee the success of our Open Gateway approach, three conditions have to be met:

- Isolation between services
- Isolation between Service Operators
- Protection of the node against both services and Service Operators

The sharing of resources in a transparent way implies that the QoS level agreed upon between the TDS and the Service Operator has to be maintained for all services at all times. That is, neither the addition or removal of services, nor their normal activity can degrade the quality or performance of other services. Furthermore, the node itself must be protected against service malfunctions or malicious implementations. The monitoring of the QoS and security levels is shared between the NodeOS and a special hardware module, called the Hardware Manager. In software, the concept of safe execution environments, sandboxes, etc. is widely known and its usefulness accepted. We believe that the ultimate responsibility of QoS and security monitoring in software can only be taken by the NodeOS. We intend to explore the

possible use of the security architecture developed inside the SwitchWare project [1] in our model.

The proposals that include programmable hardware put forward its control by the NodeOS or more specifically by the device driver. We do not think that this approach succeeds in guaranteeing isolation among services. The access to shared hardware resources, like common buses and memory blocks can not be conveniently controlled by the NodeOS, since this details are hidden from it. A greedy service could block the whole UHP by constantly sending data over the common bus or by steadily writing to memory. A malicious implementation could even try to selectively access or damage another service´s resources or data. To solve this situation we have introduced the figure of the Hardware Manager. Its main task is the monitoring of those common resources and the enforcement of QoS agreements. It basically is an enhanced hardware scheduler. Its configuration is updated by the TDS every time a new service is integrated, to take into account the new resource distribution. It is also the task of the TDS to decide, according to the existing load in the UHP, if the new requirements can be satisfied.

While most proposals do not specify how packets are directed to their respective service instances, we specifically rely on packet classification for that task. The most usual business model foresees the client requesting a certain treatment for his traffic from his Service Operator (similar to the actual SLAs). It is then the Service Operator who explicitly configures his nodes to direct the client´s traffic to the appropriate service instance. We support this model by leaving it to the Service Operator to configure the packet classification engine accordingly. Although we regard the concept of packets themselves signaling which service they require as less realistic, we do not preclude it. As stated underneath, such an approach can also be implemented inside our model.

Our proposal can be taken for a relatively conservative programmable network architecture. In reality, though, we try to present a framework inside of which a variety of network architectures can be implemented. We only limit the way in which services are integrated in the LON, in order to control the resources that it is going to use and to apply certain security measures. But we leave absolute freedom to realize any kind of service, including any other network architecture. As an example, a capsule-based approach could be realized by introducing a service which allows active packets to trigger certain functions inside this service. Our framework certainly forbids the dynamic download of new services on-the-fly. Nevertheless, most active packet approaches concede that only limited functionality can be directly transported inside the packets or downloaded on-demand and that bigger programs should be downloaded out-of-band. That fits nicely in our vision.

A safe NodeOS, the Hardware Manager and the TDS are the mechanisms which guarantee security, safety and QoS compliance in our model. We summarize the OOG´s interface structure in Fig. 5.

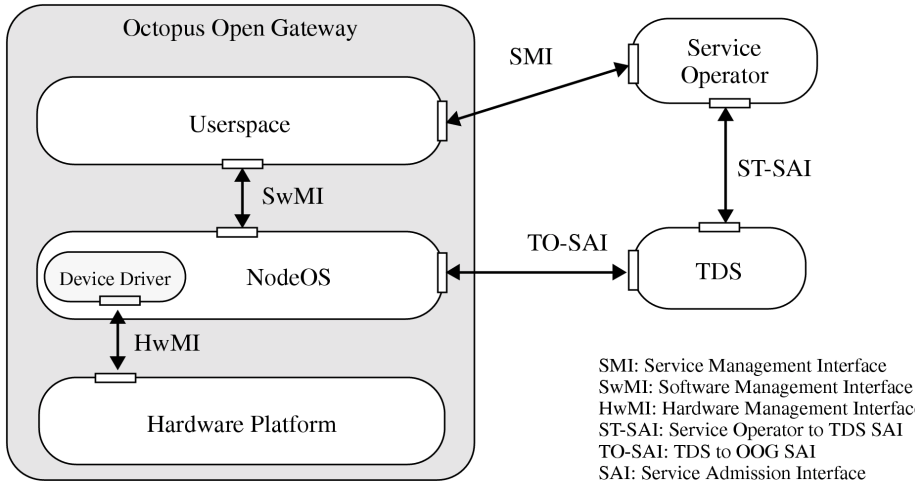

Fig. 5. The OOG's Interfaces.

The access to the network is controlled by means of the *Service Operator to TDS Service Admission Interface (ST-SAI)*. Once validated, the service is downloaded into the OOG through the *TDS to OOG Service Admission Interface (TO-SAI)*. The Service Operator can control its service through the *Service Management Interface (SMI)*. The communication between hardware and software is controlled by the *Software* and *Hardware Management Interfaces* (*SwMI* and *HwMI*, respectively). Both are supervised by the NodeOS. It falls under the responsibility of the Service Operator to implement its own SMI.

5 Conclusions and Further Work

It is commonly accepted that implementing innovative concepts in commercial networks is a difficult task. The Network Operator is mostly responsible for that situation. Technical and economic reasons discourage him to open his network to third-parties. As a result, the infrastructure as well as the management of network services is kept under the absolute control of the incumbent Operator. In order to overcome this situation so that the Active and Programmable Network paradigm can succeed, a network model is needed that:

- Preserves the security and safety of the network
- Does not degrade its performance
- Facilitates service creation, deployment and management by third-parties (Service Operators)

In this paper we introduce the Octopus Open Network Model. It represents a framework inside of which a variety of network architectures can be realized. We address the most critical points to "open up" the network by:

- Presenting a single point of entrance to service integration, where heavyweight security mechanisms can be applied, at the systems as well as at the programming level. By only leaving lightweight security checks to the node (service authentication) we also boost performance.
- Introducing programmable hardware at the core of service design to enhance performance. We also present our own vision of a Universal Hardware Platform to support these ideas.
- Substituting Network Programming Interfaces by our Service Admission Interface, in the process accepting that innovation is not foreseeable. We thus leave it to the Service Operator to define its own management interfaces.

We have also presented the architecture of the Octopus Open Gateway, which shall support our network model. Although we thankfully acknowledge the influence of previous work in our design, we introduce several innovations like the Hardware Manager, to monitor resource usage in our UHP, and the design of the UHP itself.

At the moment of writing, we are beginning the implementation of a prototype gateway. A first implementation of the UHP is almost ready. We are also working on the realization of the Hardware Manager and implementing a service that serves as proof-of-concept. We are following a hardware-software co-design approach by dividing the service in two main modules, one of them implemented in Java and the other one in VHDL.

References

1. Alexander, D. et al.: *Security in Active Networks*, in Secure Internet Programming: Security Issues for Mobile and Distributed Objects, Springer-Verlag 1999
2. Alexander, D. et al.: *The SwitchWare Active Network Architecture*, IEEE Network, special issue on Active and Programmable Networks, May/June 1998, vol. 12, no. 3
3. Campbell, A. et al.: *A Survey of Programmable Networks*, ACM Sigcomm Computer Communications review, vol. 29, no. 2, pgs. 7-24, April 1999
4. Decasper, D. et al.: *A Scalable High-Performance Active Network Node*, IEEE Network, January/February 1999
5. Decasper, D. and Plattner, B.: *Distributed Code Caching for Active Networks*, Proceedings of Infocom´98, San Francisco, April 1998
6. Feldmeier, D. et al.: *Protocol Boosters*, IEEE JSAC, vol. 16, no. 3, April 1998
7. Hadzic, I. et al.: *On-the-fly Programmable Hardware for Networks*, Proceedings of Globecom´98
8. Harbaum, T. et al.: *Hardware Support for RSVP Capable Routing*, Proceedings of the 3rd ATM Workshop, Heidelberg, June 2000
9. Psounis, K.: *Active Networks: Applications, Security, Safety and Architectures*, IEEE Communications Surveys, first quarter 1999, available at http://www.comsoc.org/pubs/surveys
10. Van der Merwe, J. et al*.: The Tempest - A Practical Framework for Network Programmability*, IEEE Network, November 1997