Trade-off or Invention: Experimental Integration of Active Networking and Programmable Networks

Stamatis Karnouskos, Hui Guo, and Thomas Becker

Abstract: **This paper discusses future requirements for enabling rapid service deployment by active networks. To enable the integration of programmable networks and active networking an object-oriented, distributed approach based on Object Request Broker (ORB) technology is presented. A Distributed Processing Environment (DPE) is described as an interoperable service platform, for dynamic and secure service provisioning support. As a platform service, a resource control framework adaptively manages router resource allocation, provides virtual resource abstraction and dynamic partitioning, to achieve the goal of scalable and secure active networking. As an example of active services, advanced reservation is developed and measured to evaluate the performance of object-oriented implementation of network services and the efficiency of underlying resource control paradigm.**

I. INTRODUCTION

Programmable Networks promote open control architectures and standard interfaces for flexible service provision to enable novel service architectures by Internet Service Vendors (ISV) [1]. Active Networks allow dynamic customization and reconfiguration of a network by means of secure code injection in it [2]. Accordingly, service modules can be encapsulated in the form of code or a composition of code fragments, and dynamically installed or updated, thereby increasing the flexibility of service deployment. In this paper, we experimented with a possible harmonization of the two approaches within a distributed object framework. Our objective was to use Active Networks mechanisms for facilitating the dynamic deployment of services, whereas using a programmable networks approach allows the deployed services to control the behavior of the hosting network elements and consequently of the whole network. The latter was achieved through the adoption of open interfaces according to the IEEE P1520 reference model [3].

In the following we summarize our design originally carried within the *Broadband Active Network Generation (BANG)* [4] project. We developed an ORB-based Active Networking (OrbAN) solution for dynamic service provisioning. The result is an active middleware framework for active networks so that active services can be dynamically deployed as downloadable objects to apply different QoS architectures on demand.

A DPE is the core of the middleware. Its key role is to manage and control resources in a secure and efficient way to support the execution of active services. This involves management of processing resources as well as communication resources both at the end-systems and network nodes. In traditional router architectures, communication resources are subject to a best effort service. Introducing programmability in legacy routers requires that vendors open-up their resources in the form of open interfaces, enabling in this way third party software vendors to deliver new and novel services.

These new services should be highly customizable and capable of binding with and controlling network resources according to the logic of the service. Such a generic router resource interface has been proposed by IEEE P1520 [5], and makes this possible by providing generic abstractions and dynamic binding capability.

On the other hand, as services in an active network share the resources with each other during run-time, their access needs to be synchronized and policed in order to avoid conflicts and to provide service isolation. To optimally utilize the node resources, more intelligent allocation of resources is required than fixed partitioning.

In the following, we describe the active node architecture and then the binding and resource control frameworks with focus on dynamic partitioning. Subsequently we present the dynamic deployment of active code framework as well as the reservation in advance. Finally, we measure and comment in order to provide insight on the performance of ORB-based Active Networking.

II. ACTIVE NODE ARCHITECTURE

In our approach we address the trade-off between flexibility and performance, which is currently one of the most important problems the Active Network community is facing. It is argued that Active Network platforms generally sacrifice significant performance to gain flexibility. Thus, our driving force was to point out where it is worth trading-off performance for flexibility in network management or even to find ways to achieve flexibility without performance loss. Exploiting the achieved network flexibility and dynamics, we try to solve existing problems that cannot be easily solved in the traditional networking paradigm. Fig. 1 shows a view on the ORB-based node architecture and the mapping to the P1520 interface levels.

In the architecture we have the following three levels:

1. **Fixed part (router hardware):** This part contains static and optimized forwarding components which cannot be made programmable due to performance reasons. The fixed part follows the current trends of forwarding technologies where switches and increasingly high perfor-

The authors are with German National Research Center for Information Technology Research Institute for Open Communication Systems GMD-FOKUS, Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany, e-mail: karnouskos@fokus.gmd.de, guo@fokus.gmd.de, and becker@fokus.gmd.de.

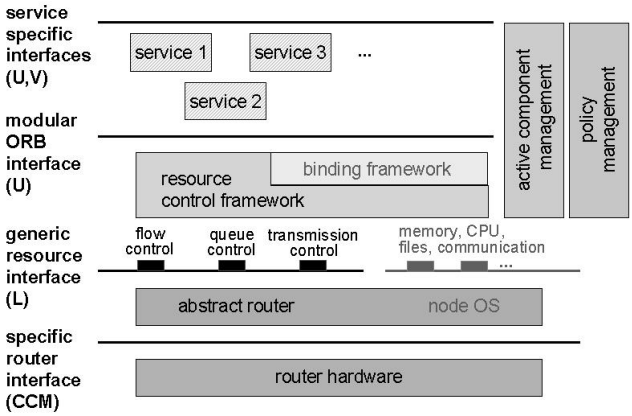

Fig. 1. Architecture of ORB-based Active Networking.

mance routers implement their forwarding functionality in hardware and avoid adding software in the performance-critical forwarding path. In this architecture, *user packets* which make up for the majority of packets flowing through an active router are processed and forwarded directly by hardware in the fixed part.

2. **Programmable part (abstract router plus resource control framework):** This part encompasses the specific or proprietary interfaces of the fixed part and provides the higher level with an open standardized interface. Exploiting the features and high performance of the fixed part, the programmable part typically implements network algorithms to provide end-to-end services. In particular, in order to provide a good *perceived quality*, multimedia applications typically require that certain end-to-end quality of service (QoS) is guaranteed. The programmable part can fulfil the requirements of quality of service by implementing current network algorithms like RSVP/IntServ and/or DiffServ with some necessary extensions to allow for the programmable access of the high-level active part. These networking algorithms are implemented as programmable modules on top of the fixed part.
3. **Active part (active services):** This part offers a *limited* execution environment for *lightweight active components*. Lightweight active components contain either generic or application-specific algorithms needed for value-added IP and high-level services. Because these algorithms are typically application-specific, they are not implemented together with the generic network algorithms of the programmable part to avoid unnecessary code bloating. Lightweight active codes use the module interfaces of the programmable part to access the functionality implemented in hardware of the fixed part. Lightweight active code typically contains some function calls or simple scripts to the module interfaces of the programmable part with specific parameters. In general, the active code cannot be generic since it has to meet the specific requirements of the applications (e.g., application-specific drop policy or queue management).

The DPE is built on top of a modular ORB framework [6]. It is active in two ways:

1. it is running on active nodes supporting active services,
2. it provides transparent access to active services for applications by encapsulating the interaction with services inside the (client-server) bindings of application objects.

The modular ORB framework mainly consists of a binding framework and a resource control framework where the former is responsible for giving access to remote objects and the latter to resources. The resource control framework was extended by a resource manager for router resources, which offers interfaces to active services. The active services can then make use of resources in order to provide a higher level service (via P1520 U-interfaces) to value added services or directly to applications.

Active services are deployed to active nodes by the Active Component Manager (ACM) who is acting as a bootstrap service and manages the lifecycle of service components. The resource allocation for service components is done through resource managers[1]. The deployment of components and the resource usage is guarded by policies. The access to the functionality of the generic router API is managed by a router resource manager.

The three-level architecture we follow achieves the necessary flexibility in active networking without losing the router's performance: user packets which make up for the majority of the traffic flowing through an active router are processed and forwarded directly by hardware in the fixed part. Control packets without application-specific requirements are serviced by generic network mechanisms in the programmable part. Finally, the active part allows users' active code to be executed in *limited* execution environments in order to obtain high-level and value-added services. This node architecture is very interesting for vendors providing legacy switches or routers, since their proprietary interfaces or even functionality can be wrapped by a generic interface and therefore easily integrated into an active network environment. Thus, via our approach we are calling not for a revolution in networking technology but rather for gradual evolutionary steps from traditional networks via programmable ones and finally to active networks.

III. BINDING FRAMEWORK

Communication between objects supported by the framework is through bindings, which are created by *object adapters* known from the CORBA architecture [7]. In this framework, the notion of object adapter is overloaded and extended to allow the explicit binding of objects: the explicit creation of a binding between different interfaces is realized by invoking an operation on an object adapter. Object adapters are *binding factories*.

In contrast to the CORBA architecture which identifies an ORB core responsible for the conveyance of operation requests and replies, the notion of object adapter in this framework is extended to cover also communication aspects, which may thus vary from object adapter to object adapter. In summary, an object adapter is not limited to cover the server side as in the standard CORBA specification, but actually is extended to the client side. The notion of ORB core in CORBA can be recovered as a

[1] Conceptually all kinds of resources (e.g., memory, CPU time, and router resources) are managed by the resource control framework. The focus of the implementation is on the router resource management.

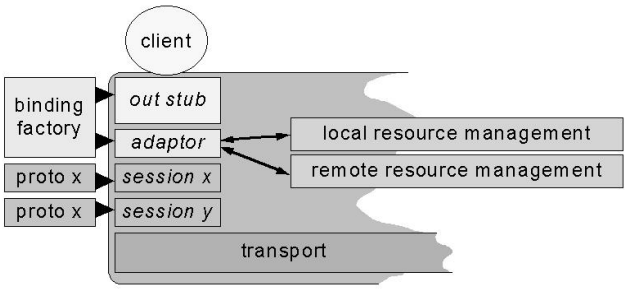

Fig. 2. Architecture of a binding (client side).

specific, default object adapter that can be combined with other object adapters.

This flexibility allows us to define a special binding factory which understands additional parameters like QoS requirements for the creation of a binding. This binding can then provide an interface to application objects to allow dynamic changes of its behavior as well as offering registration for notifications about status changes.

The binding framework consists of a set of abstractions for the construction of arbitrary communication stacks and abstractions for the construction of protocol-independent operational stubs. Communication abstractions comprise:

- *Protocols:* these are abstractions of protocol machines at a given site; they manage the establishment and release of sessions.
- *Sessions:* these are logical communication channels that obey a particular communication protocol; sessions in different capsules exchange messages.
- *Messages:* these are abstractions of data exchanged between capsules.

Protocol-independent stubs described in the binding framework provide generic interfaces for operational bindings. They can be specialized to derive more specific forms of operational bindings. Stubs are at the interfaces between the untyped world of protocols and the typed world of language bindings.

The open architecture of the binding framework allows the easy insertion of components into the communication stack (see Fig. 2). For the support of QoS the binding factory deploys a special adaptor on top of the session stack. The adaptor interacts with the local and remote resource management.

The interaction with the local resource management comprises contacting resource managers for processing resources also known as schedulers, managers for memory, and managers for local network interfaces. The interaction with remote resource management is achieved by negotiating QoS with intermediate network nodes and the target end-systems of the binding. To achieve end-to-end QoS for an object binding, the binding has to interact with both local and remote resource management.

IV. RESOURCE CONTROL FRAMEWORK

One of the most distinct advantages of active networks is the dynamic service provisioning. More specifically, service code can be dynamically downloaded, installed and instantiated. Service instances are thus created and executed where and when they are needed. Their lifecycle may vary, contrary to that of today's network services which is fixed. This requires that the platform or the service environment that supports their execution, reacts more proactively on their continuously changing need for resources. On the other hand, service instances may impose different control architectures, e.g. heterogeneous QoS paradigms, for the exploitation of the physical network resources. A service platform for active networks should be able to accommodate this heterogeneity in such a way that the service environments are kept isolated and secure from each other while the performance compromises are minimal.

During the BANG project, a flexible service platform (i.e., the DPE) for active IP networks was developed by enhancing the modular ORB. To fulfill these critical requirements, a router resource manager (RRM) was implemented as part of the resource control framework in order to provide:

- on time, efficient and safe resource allocation for active services, and
- isolated execution of active services.

On time allocation helps to build a platform, which can adaptively meet services' needs. It refers to the capability of fast reaction on the changes of the active services' resource needs, or the current network load. This can be achieved either by pipelining the allocation operations, or proactively detecting the changes and adapting the allocation autonomously without the intervention of services.

Safety of allocation means non-violation of the Service Level Agreement (SLA), which specifies the parameters regulating an allocation. Basically a resource management paradigm should deny the allocation requests from unauthorized services, and guarantee that a service will get the amount of resources on time as specified in the SLA. A SLA is normally negotiated between a service provider/customer and a network provider and defines the amount and the type of resources his/her services will need on a physical network.

Efficient allocation is extremely important in order to deliver a scalable service platform and to maximize the usage of limited network capacity. Mainly active network providers are the actors who benefit from this capability. Pure SLA-based allocation paradigm is naturally rigid, and loses the multiplexing benefit. In order to achieve high efficiency, a dynamic bandwidth allocation paradigm is needed, which was one of the major points of focus within the BANG project.

Isolation of execution (sandboxing idea) provides a basis to protect active services from each other. The security manager-based approach is restrictive by mandating that a service environment is needed for each service instance, which implies overhead. In addition, it only supports Boolean restriction, i.e., Yes/No for controlling resource access. We need a more flexible scheme allowing a finer granular access control with a set of quantitative measures.

RRM contains the following key modules: *Policy Controller, Flow Manager, Partition Manager, and Admin Manager*.

The *Policy Controller* maintains the policies used for controlling the resource access and partitioning of network resources. The *Flow Manager* provides virtual resource abstractions for

ServicePolicy:

ProviderID	ServiceType	ResourceType	NIF	Quota
ISP1	ReA	GoldCLS Q...	Ethernet4	20%
ISP2	ReA	GoldCLS Q...	Ethernet4	80%

ResourcePolicy:

NIF	ResourceType	Quota	Link Capacity
Ethernet4	GoldCLS Queue	80%	100Mbps

Fig. 3. Policy example.

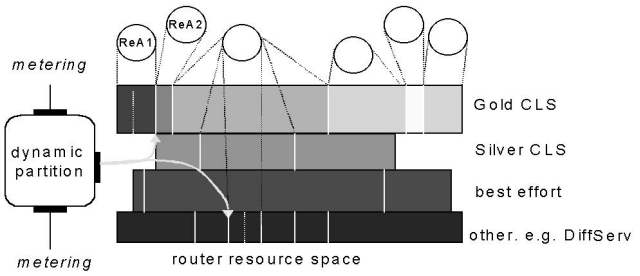

Fig. 4. Dynamic resource partitioning.

active services and makes admission control to enforce safety. The *Partition Manager* implements a dynamic partitioning algorithm to adaptively adjust the policies and enforce them on runtime. The *Admin Manager* serves as a portal for network administrators to monitor and tune the router resource manager parameters. A core scheduling module called *scheduler* synchronizes their execution and arranges interaction to guarantee the integrity of the router resource manager in the whole OrbAN system.

Physical router resources are partitioned into a set of virtual resources, e.g., a network interface card consists of 8 virtual output queues. These virtual resources are of different type, e.g., a virtual queue can be called GoldCLS_Queue, which means it provides a gold controlled load service for flows. A resource policy is thus defined for each type of virtual resource to regulate how much of this physical resource is mapped to it. For instance a GoldCLS_Queue can be mapped to a physical queue on a 100 Mbps Ethernet card No. 5 with priority 8 and capacity 80 Mbps. An active service instance has access to a single or multiple type/instances of virtual resources; a service policy is defined for each of them. The policy specifies "which active service instance is allowed to access how much of which type/instance of the virtual resource." Therefore it defines the resource limit that an active service instance can have exclusive access on. The *Flow Manager* uses this value to make admission control to ensure safety and minimal interference. These policies are maintained in *Policy Controller* and can be updated by a network administrator at any time. It is his/her task to ensure these policies are consistent. Through resource virtualization and policing, the goals of secured allocation can be achieved. Some examples of the policies are depicted in Fig. 3.

The *Partition Manager* aims at providing a more efficient allocation by intelligently estimating the current resource utilization ratio and future resource needs of all running active service instances as depicted in Fig. 4.

An Active Service Instance (ASI) represents a network service for a particular service provider. Different ASIs might have different policies for provision, though possibly with same code. Multiple ASIs run in parallel and access router via the flow manager of a RRM. Each ASI is allocated with a quota as its resource limit is defined in a service policy, i.e., it can only manipulate (re-allocate, merge, release, etc.) this resource for its users. The *Partition Manager* dynamically examines the traffic load and the usage of the quota, and runs an intelligent algorithm to determine the new quotas. By assigning new quotas, those ASIs that have very high resource utilization will get more resource quota to accommodate more requests from their users; others with very low resource utilization will have less resource quota. So this is a pre-emptive model which means "busy/starving ASIs get more quota while free/stuffed ASIs get less." This model is similar to that whereby high-priority thread pre-empts low-priority thread, but it is not restricted by pre-defined priorities [8].

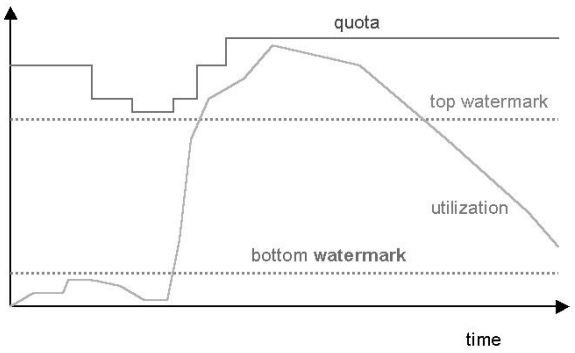

Fig. 5. Dynamic quota adjustment.

For this purpose, a *Flow Manager* provides a meter interface for estimating current resource usage of each ASI; while an ASI might provide a call back meter interface for estimating future resource need. The algorithm for calculating new quotas is depicted in Fig. 5.

In short, the utilization ratio of each active service instance is checked against two thresholds called *Top Watermark* and *Bottom Watermark*. If it is smaller than *Bottom Watermark*, current quota will be decreased by a certain basic unit. This is called *Quota Releasing*. The quotas released in this way forms a *quota pool* representing free resources. If the utilization value is larger than *Top Watermark*, current quota is increased by a certain basic unit if the *quota pool* is not empty. This is called *Quota Allocation*. By more adaptively changing the resource limit according to the need of each ASI, the ASI is able to accommodate more user requests and achieve higher utilization.

V. DYNAMIC ACTIVE CODE DEPLOYMENT

The opportunity offered by active networks to dynamically install components for execution in network nodes offers a high degree of flexibility and several other advantages to network management. On the other hand, such an action exposes a serious security issue: malicious or bad designed components could damage or cause malfunctions to the active nodes. In order to tackle these drawbacks, our design is based on the following rationale: a) Active components are installed via a policy-

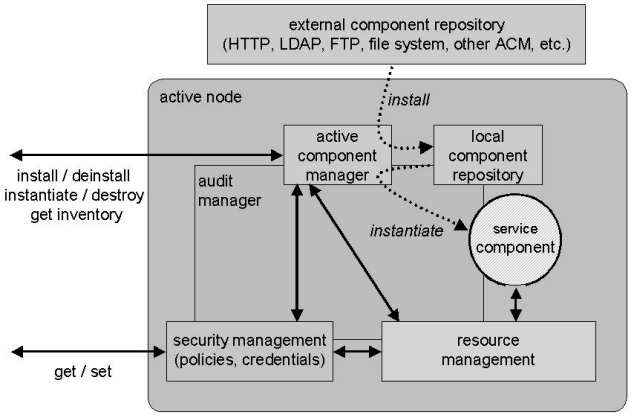

Fig. 6. Active component management service architecture.

controlled way from internal or external repositories, b) policies are defined for the resource usage and allowed behavior of a component and the overall system, and c) the security manager is consulted for all security critical activities.

Active Component (AC) is a service component that executes within an Execution Environment (EE) in an active node. An Active Component can maintain its state from node to node transition, or be stateless (no state is maintained). It could be itself mobile (e.g., an intelligent mobile agent) or could be transferred to the active node by other third entities. The Active Component Manager (ACM) allows AN entities (e.g., users, administrators, etc.) to install AC on the node and make use of it or possibly make it available to other third party entities via a policy controlled way.

In the ACM architecture depicted in Fig. 6, we can identify the following major components:

Active Component Manager: This is the front-end of the architecture. All requests are issued to, scheduled and executed or denied by this component. Other system and service components stored in ACM's DB are loaded and instantiated by the ACM.

Security Manager: This component is responsible for all security relevant activities. It takes security decisions and grants or denies the issued requests. Checks are made to ensure that i) only authorized users install and interact with node's services and ii) the policy of resource usage by the installed components is enforced. The policies for component/service access are maintained also by this component. Via its interface, authorized entities can dynamically modify the policies in the EEs or those in the node.

Audit Manager: All events are audited by this component for further exploitation.

Resource Manager: This module controls the allocation and access of the local node resources (computing resources and network resources). The access to resources is controlled in cooperation with the Security Manager.

ACM Repository: It is the repository that AC is stored. This is local to every node and also provides interfaces so that it can interact with other ACM code repositories in other nodes. If code is not local in the node, it is fetched via the usage of well-known protocols such as https, ftps, ldaps etc. The ACM repository may also contain references to the components it has although they are not present at the local DB at the moment.

A possible scenario that shows the interaction between the various ACM parts is as follows:

1. *Request:* A request is made to the ACM to install a component/service. The request might be issued explicitly by a user (the user is generally any authority—the difference is depicted via the policy scheme with the use of access rights) or implicitly as a side-effect of the setup of an object binding requiring a certain service on the active node.
2. *Security check:* The ACM consults the Security Manager (SM) whether the specified action is allowed or not. The SM verifies the credentials of the authority that issued the request and then checks the current policy. The Resource Manager (RM) is also consulted whether the action complies with the resource limits. Finally the SM returns an accept or deny result for the specified action.
3. *Process of Request:* The ACM executes or denies the user request. E.g., installation, deinstallation, instantiation, destruction, service start, service stop, AC retrieval, service/code search, etc.

The actions following the last step vary as they depend on the nature of the request issued. We can have:

- *Download:* if the request is valid and the components are not cached locally, the service contacts another repository (e.g., via https, ldaps, etc.) to download the requested component.
- *Resource allocation:* after the component is downloaded the appropriate resources are allocated (i.e., a new job is created to run the tasks of the component).
- *Instantiation:* the component instantiates and executes in a policy-controlled environment.
- *Runtime checks:* all interactions of the installed component with the resource management are checked with the policy management, this ensures that the component does not exceed its predefined amount of resource usage, nor it violates the given access rights.

VI. RESERVATION-IN-ADVANCE ACTIVE SERVICE

This section aims to describe a scenario which demonstrates the advantages of the active DPE. For this purpose a generic resource reservation service is sketched as one active service that can be customized for different styles, e.g., reservation-in-advance [9] or immediate reservation (RSVP).

Fig. 7 shows a distributed reservation service. This service consists of components sitting on top of the resource management on each active node. With the help of chaining components on several nodes a network-wide service can be offered to applications. In the end-systems the service interfaces are accessed out of the binding framework. A special proxy object (denoted Q in the figure) handles the QoS needs of the application objects and interacts with the local resource management as well as with the distributed reservation service.

The components forming the reservation service are stored in a trusted repository managed by a network operator. In the deployment stage those components are downloaded and installed

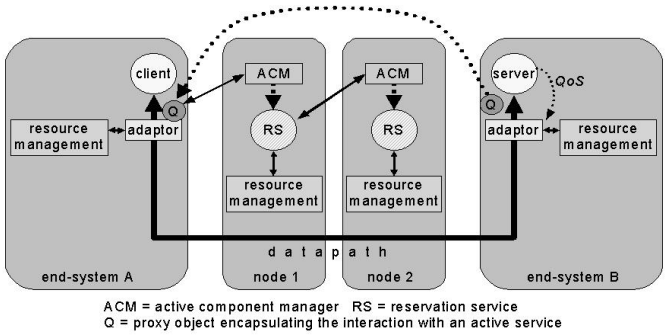

Fig. 7. Distributed reservation service for object communication.

on the active node by the installation service. The run-time instance of a reservation service component has its limited resource space, allocated by the installation service and controlled by resource managers.

To provide a network-wide interface the service instances on different nodes have to interact. For this, an instance has to be able to obtain the interface references of other instances in neighboring active node. This can be achieved by a centralized CORBA naming service [10], or a propagation protocol among active nodes that makes the references aware to adjacent nodes. The way the chain of service components is build is specific to the implementation of the service and not part of the framework.

The QoS expected for the communication between objects can be specified when the binding is created. Server objects may export their interfaces to the binding specifying the QoS they expect at their interfaces, client objects may import the interfaces also specifying the QoS they expect for the communication. In any case the QoS has to be established along the communication path between client and server. This can happen when a client imports an interface, i.e., connects to the binding, or on the first call on an imported interface.

The request for establishing a QoS has to be propagated along the communication path and each node has to decide whether the request can be fulfilled. Following the admit/reserve pattern described in a previous chapter it can be avoided to reserve resources without knowing if the reservation is admittable on all intermediate network nodes. Of course one has to take care about network nodes along the communication path that don't support the distributed reservation service. This problem can be solved by over-provisioning or by adapting to available reservation techniques.

The purpose of the distributed reservation service is to provide resource reservation for the communication between a multitude of distributed objects. The main objective is to share the available resources between requesting applications as effectively as possible. For this additional information like priority policies or timetables could be useful. The dynamic deployment of the service components allows a flexible response to the needs of applications.

VII. IMPLEMENTATION

A major goal of the presented architecture is to support different router hardware—especially legacy systems—and allow a variety of active services to be deployed to an active node. This is achieved by implementing a generic router API which is wrapping the specific command interface of a router and provides a common way for accessing the router's resources.

To obtain a high flexibility and portability the generic router API, as well as active services using this API, are implemented in JAVA. However, the fast forwarding of standard IP packets is retained since the forwarding path of the router is in general not intercepted but controlled.

The main difficulty with several active services running in the same JAVA virtual machine, is the true isolation between them with regard of memory and processing resources. This could be achieved by using the JAVA profiling interface [11] or a specially tailored JAVA virtual machine [12].

Access to resources is controlled by resource managers where policies are used to define the allowed resource usage for particular identities. The policies are implemented by JAVA objects stored in a database, offering an interface to administrators to set, get, and modify them. The objects implementing active services are also stored in a database along with some information about the version and current status. This allows an administrator to manage the available services on a node.

A prototype of the active DPE is implemented in JAVA using the modular and extensible Jonathan ORB [6] as the basis for the binding and resource control frameworks. The active DPE is deployed in a testbed consisting of three Hitachi GigabitRouter 2000 [13] and three controlling PCs running on Linux. The active DPE is running in a JAVA virtual machine on the controlling PCs and accesses the router command interface via a Telnet connection. The router's command interface if wrapped by JAVA objects forming a generic router API. Currently the DPE doesn't support packet processing, it only features the management of router resources. All interfaces of this implementation are written in OMG's Interface Description Language (IDL).

VIII. MEASUREMENTS

A. RRM Measurements

The measurements were based on a simple methodology. Namely, checkpoints are placed in the RRM functions such as dynamic partitioning module, flow management module, and policy control module. They are triggered by a timing service, and print out the core states of RRM with regard to the current resource utilization ratio of each active service instance, and virtual queue. As the measurement was integrated into the prototype, many factors affect the results and make it difficult to interpret them. For instance, real-time thread scheduling in Java virtual machine cannot be controlled, and garbage collection affects the time consumed for processing reservation requests and making allocations. Also, control of timing scale is limited by Java and this has also an impact on the normal execution of active service instances. In addition, simulation of the request arrival model is hard to implement and replicate.

Therefore the measurement only provides a partial proof-of-the-concept for our approach. The average resource utilization and admission percentage are the major results measures to evaluate how RRM improves the resource usage by dynamic partitioning.

In the following we present the measurements of the performance benefits brought by dynamic partitioning, as compared to static partitioning for management of router resources. We use the advanced reservation service (ReA) [8] as one example of active service, and run two ReA instances for different service providers.

The parameters tuned during the measurement are:

- **Time to measure:** 10 minutes.
- **Partition scheme:**
  - *Top watermark:* threshold for max. Resource utilization ratio; set to 75%.
  - *Bottom watermark:* threshold for min. resource utilization ratio; set to 15%.
  - *Incremental unit:* basic unit to increase/decrease the quotas; set to 6%.
  - *Partition cycle:* 6 seconds.
- **Active service:**
  - *Service pattern:* the arrival model of the end-user reservation requests, random distribution and gauss distribution are used; duration of each reservation is set to 2 minutes.
  - *Traffic model:* the traffic characteristic of flows; set to a Constant Bit Rate with 100 kbps per flow.
  - *Number of requests:* random service pattern—for ReA of Internet Service Provider 1 (ISP1), 300 requests, for ReA of ISP2, 50 requests arriving at the same time; gauss service pattern - for both ReAs, 300 requests arriving simultaneously.
- **Policies:**
  - *Service policy:* for ReA of ISP1, quota = 20% of GoldCLS_Queue; for ReA of ISP2, quota = 80% of GoldCLS_Queue.
  - *Resource policy:* for GoldCLS_Queue, quota = 80% of Link capacity.
  - *Link capacity:* 100 Mbps for Ethernet.

The parameters to be measured and analyzed are:

- **Utilization ratios:** for ReA of ISP1, for ReA of ISP2, for GoldCLS_Queue.
- **Quotas:** for ReA of ISP1, and ReA of ISP2.
- **Rejection percentage:** percentage of reservation requests rejected due to unavailability of resources, for ReA of ISP1 and ReA of ISP2.

Figs. 8 and 9 depict the results. As these graphs show, for gauss service pattern, the admission percentage increases from about 45% (static partitioning) to 60% (dynamic partitioning). The average utilization of GoldCLS_Queue increases from about 30% to 60%.

B. ACM Measurements

ACM is responsible for installation/deinstallation and instantiation/deinstantiation of active components among other things. We wrote a test suite to measure the performance of ACM. All tests took place in a PC with one Intel Pentium III 500 MHz processor, 128 MB RAM running Debian Linux 2.4. The results are depicted in Fig. 10.

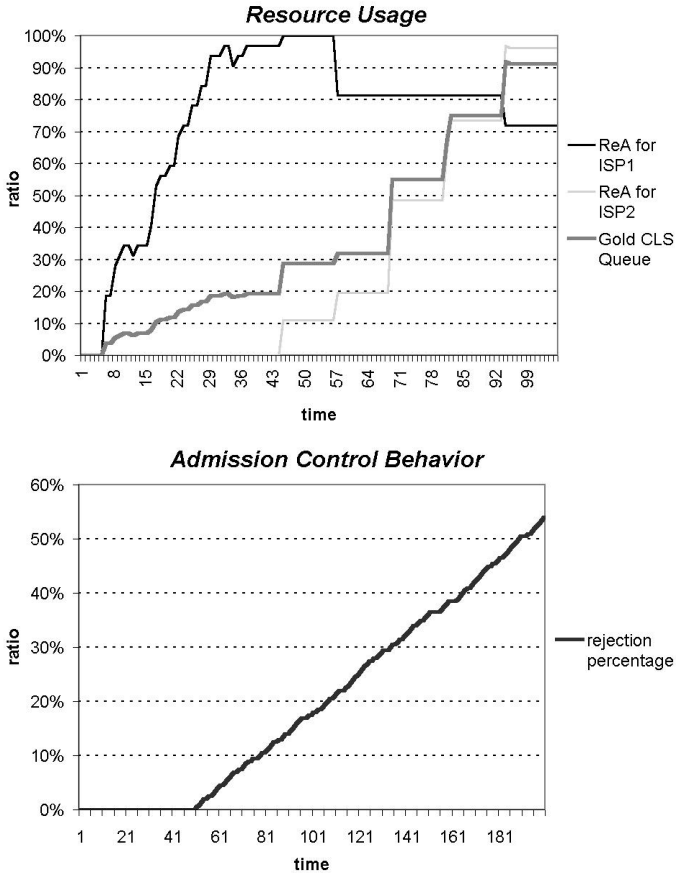

Fig. 8. Reservation-in-advance service with static partitioning (Gauss service pattern).

We have measured the time it takes to install and deinstall a number of active components as well as the time to instantiate and deinstantiate the active component. The instantiated active component code represents a service running on the active node. All measurements of ACM above represent successive operations on a total number of requests on the system. As expected the initialization of the objects (installation and instantiation phase) is more time consuming than their destruction.

Both diagrams above were expected to be linear or at least linear-like. However the tests resulted to something else, including some peaks that we cannot explain with certainty. These are probably a side effect of the cooperating components and the network status. During the installation phase, the component has to be fetched and therefore the network utilization at that time can have an effect on the measurement. Also during instantiation, the new service that the component implements has to register with the central naming service that relies on another computer. For the deinstallation phase, we first search our database for the specific component and when we find its position we remove it. Therefore the size of the database and the implementation of the search function can affect the time consumed for this action. Also during deinstantiation phase the service (that the component implements) has to be stopped (if active) and deregistered from the central naming service (network side-effects).

In addition to these, the Java garbage collector activities inter-

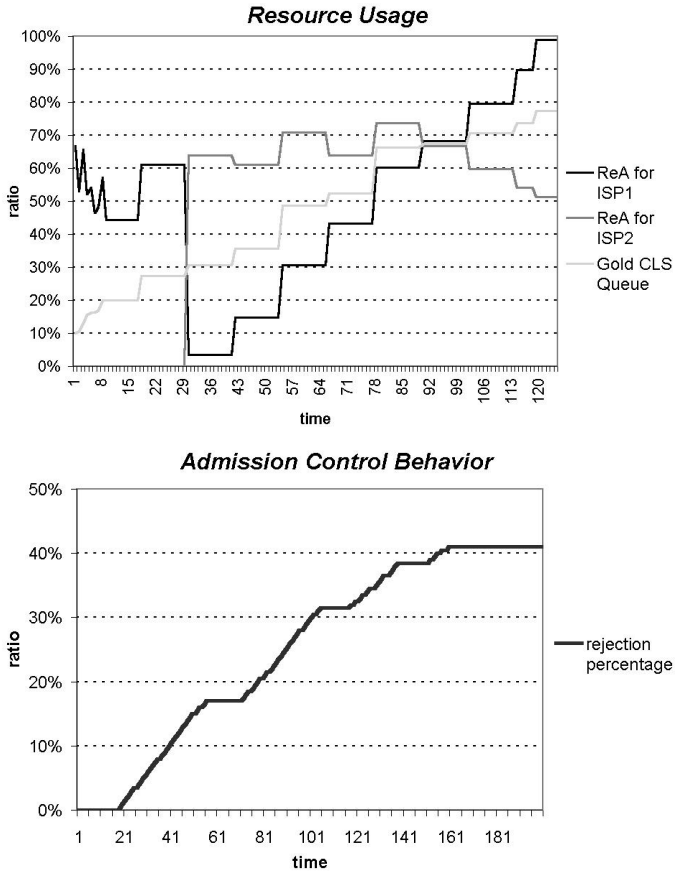

Fig. 9. Reservation-in-advance service with dynamic partitioning (Gauss service pattern).

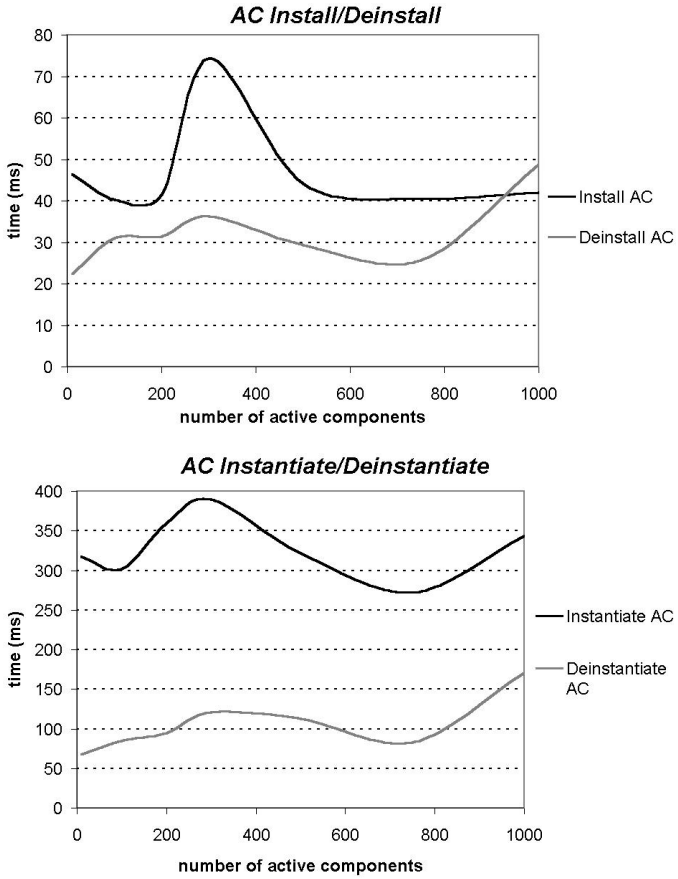

Fig. 10. Performance of the active component manager.

fere with the measurement process, since we create and release several hundreds of Java objects in the test suite. All above actions are done also via a policy controlled way, therefore include the time of reading the policies, of checking the vector of policies in order to apply the right ones, of checking the provided credentials, etc. The algorithms used for the above functions are not optimized and therefore we consider that these measurements are very near to the level of the most time consuming ones we can get. By implementing more intelligent algorithms and optimizing the Java source code of the OrbAN implementation, in our opinion the times will decrease drastically. However even with this novel approach we have acceptable results regarding the active code management. Even with the above mentioned side-effects, the installation of the active code remains between the 40 ms and 80 ms zone, and its deinstallation between 20 ms and 45 ms zone. For active code instatiation the time is between 280 ms and 400 ms, and for deinstantiation between 90 ms and 180 ms. To our opinion these non optimized time measurements are generally acceptable for a maximum of 1000 successive requests (this is considered as a normal operation utilization) in our active node.

IX. CONCLUSIONS AND FUTURE WORK

A framework for an active DPE was presented to facilitate the integration of programmable networks, active networks, and distributed object technology. The presented approach solves the problem of performance vs. flexibility by offering a three level architecture supporting high performance in the fixed part and a high flexibility in the active part.

The framework supports the execution of network services as downloadable active components. The task of these services is to provide QoS to applications by flexibly programming network resources through a dynamic and efficient resource manager interface based on the P1520 L-interface [14].

Their parallel execution with respect to resource access and usage is monitored and controlled in order to ensure safety. Basic measurements show the efficiency of resource management, and the flexibility of customization via policies.

Up to now, most research found in literature only provided stand-alone solutions, i.e., an active network approach with code distribution/loading in the core, or programmable networks supporting open service implementation. For instance, ALIEN [15] defines a layered architecture to provide fine-grained control of code loading. It focuses on the core functions such as *core switchlet* in an active node, instead of the underlying network abstractions and it does provide a clear computing model as a guideline for us regarding the core node functions. Elastic Network Control [16] is more relevant to our approach. It enhances traditional control/management architecture by virtual networking and partitioning, and enables dynamic deployment of new control paradigms. However it is based on ATM switches that provide standard support for virtualized network abstractions, e.g., VPI/VCI.

The challenge in IP networks is that no standard or product exists that can deliver the level and granularity of resource abstractions needed to program the network. Also a more efficient usage of limited resources demands a flexible multiplexing paradigm like dynamic partitioning. From the resource control perspective, RCANE [17] developed QoS management using active network systems, but focused on the control of computing resources such as CPU time memory, and network I/O in the operating system. In addition to that we believe control of access to the router resources is essential in order to support secure and efficient execution of new active services. SANE [18] relies upon the proprietary multiprocessor operating system Piglet in order to offer a resource management interface, while our focus is on the open API for network programming. In addition we deploy operator-defined policies to control the access, which is more capable to cover the various requirements, e.g., time scale, than mere credentials.

Our approach tries to allow deployment of a variety of active network services independent of the underlying router hardware infrastructure. Via a generic router API and the wrapping of its functions we aim also at integrating current legacy systems. In the future we will further extent our interfaces to be fully compatible with the P1520 work and we will compare our system with other ones in matters of performance and operation times.

REFERENCES

[1] A. Lazar, "Programming telecommunication networks," *IEEE Network*, Sept./Oct. p. 818, 1997.

[2] D. L. Tennenhouse *et al.*, "A survey of active network research," *IEEE Commun. Mag.*, pp. 80–86, Jan. 1997.

[3] J. Biswas *et al.*, "The IEEE P1520 standards initiative for programmable network interface," *IEEE Commun. Mag.*, Oct. 1998.

[4] Broadband Active Network Generation (BANG), a Hitachi and GMD-FOKUS Co-operation Project. Available at http://www.fokus.gmd.de/research/cc/glone/projects/bang/.

[5] IEEE P1520 Project. Available at http://www.ieee-pin.org/.

[6] B. Dumant *et al.*, "Jonathan: An open distributed processing environment in Java," Sept. 1998. Available at http://www.objectweb.org/jonathan/current/Jonathan.html.

[7] "The common object request broker: Architecture and specification," revision2.4: Oct. 2000. Available at http://www.omg.org/technology/documents/formal/corbaiiop.htm.

[8] T. Becker, H. Guo, and S. Karnouskos, "Enable QoS for distributed object system with ORB-based active networking," in *Proc. 2nd International Conf. Active Networks (IWAN'00)*, Tokyo, Japan, Oct. 2000.

[9] A. Schill, F. Breiter, and S. Kuhn, "Design and evaluation of an advance reservation protocol on top of RSVP," in *Proc. IFIP 4th International Conf. Broadband Communications*, Stuttgart, Chapman & Hall, 1998, pp. 23–40.

[10] "Interoperable naming service specification," new ed., Nov. 2000. Available at ftp://ftp.omg.org/pub/docs/formal/00-11-01.pdf.

[11] The Java Virtual Machine Profiler Interface. Available at http://java.sun.com/j2se/1.3/docs/guide/jvmpi/index.html.

[12] The Java-oriented Active Network Operating System (Janos). Available at http://www.cs.utah.edu/flux/janos.

[13] " Hitachi GR2000 Gigabit routers . " Available at http : //www . internetworking.hitachi.com/products/products_GR.html.

[14] IEEE-P1520 L Interface Specification draft. Available at http: //www.ieee-pin.org/doc/draft_docs/IP/p1520tsip012-1.pdf.

[15] D. S. Alexander and J. M. Smith, "The architecture of ALIEN," in *Proc. 1st International Working Conf. Active Networks (IWAN'99)*, June/July 1999.

[16] Herbert Bos., "Open extensible network control," *J. Network and Systems Management*, vol. 8, no. 1, Mar. 2000.

[17] P. Menage, "RCANE: A resource controller framework for active network services," in *Proc. 1st International Working Conf. Active Networks (IWAN'99)*, June/July 1999.

[18] D. S. Alexander *et al.*, "Secure quality of service handling," *IEEE Commun. Mag.*, vol. 38, no. 4, pp. 106–112, Apr. 2000.