Enable QoS for Distributed Object Applications by ORB-based Active Networking

Thomas Becker, Hui Guo, Stamatis Karnouskos

German National Research Center for Information Technology

Research Institute for Open Communication Systems

GMD-FOKUS, Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany

{becker, guo, karnouskos}@fokus.gmd.de

Abstract. Future requirements for supporting distributed object applications by active networks are discussed. A middleware approach for active networking is presented. A CORBA-based distributed processing environment (DPE) is described as an interoperable service platform in an active network to enable end-to-end QoS for distributed object communication. Three key platform services are presented, including QoS binding, component management and resource control. A binding framework is enhanced to achieve transparent QoS binding; an active component management service is proposed as an out-band signaling to install service objects; active node resources are adaptively managed to support generic reservation requirements. As a whole, the paper presents a distributed computing model for active networks so that active services can be dynamically deployed as downloadable objects to apply different QoS architectures on demand. With this model, distributed object-oriented systems directly benefit from active networking technology with respect to QoS need.

1 Introduction

Programmable Networks [6] promote open architectures and standard interfaces for flexible service provision to enable novel service architectures by Internet Service Vendors (ISV). Active Networks [18] allow dynamic customization and reconfiguration of a network by means of secure code injection in a network. As a reconfiguration example, service modules can be encapsulated in the form of code or a composition of codes, and dynamically installed or updated. Therefore, it greatly increases the flexibility of service deployment. Harmonization of the two approaches within a distributed object framework therefore facilitates deployment of services, either application-specific or generic, to better support distributed object applications than today, in terms of, e.g. Quality of Service (QoS). The framework can be seen as an interoperable (e.g. by IIOP) distributed platform for running active services. This paper presents a technical approach to show how active networking can be used to

improve the execution of distributed applications, with the help of a middleware bridge.

Providing end-to-end QoS for distributed object systems is the major focus of our approach. In existing frameworks such as CORBA, providing QoS for the communication between distributed objects is challenging due to the difficulty of deploying a generic QoS architecture [7], which might require an integration of appropriate architectures such as Integrated Service[15] and Differentiated Service [10] across the network to obtain an end-to-end QoS guarantee. On the other hand, in many cases the QoS requirements from application objects cannot be mapped or translated into network QoS parameters as supported by different protocols, e.g. RSVP [16]. Thus QoS solutions [1] in the middleware are normally relying on and closely coupled with particular network-layer QoS mechanisms. Related work includes Active Reservation Protocol [3] which enables portable signaling software but is bound to Java and therefore limited by Java security and interoperability and Xbind [12], which uses CORBA as the platform for programmable value-added services but is limited by existing ORB implementation with regard to QoS support.

In the following the paper summarizes our design in the "Broadband Active Next Generation" [4] project. We developed an active middleware framework that supports end-to-end QoS for a wide range of distributed applications. It represents a distributed computing model for active networks so that active services can be dynamically deployed as downloadable objects to apply different QoS architectures on demand. The framework mainly refers to an active distributed processing environment (DPE) that is an execution environment (EE) based on an enhanced Object Request Broker (ORB).

In general, the role of a DPE is to ease the development of distributed applications. An application object can access the interface of other objects without knowing the location of those potentially remote objects. The DPE is used to gain access to interfaces, that is to set up a communication path between objects. The ODP Reference Model [13] defines a generic model for distributed processing and standards as CORBA [8] specify a concrete architecture supporting this. Since CORBA is limited in the types of object communication it supports, the more open Jonathan architecture developed during the ReTINA project [9][11] is used as a basis.

Another important role of the DPE is to manage resources in a distributed way so that an end-to-end QoS can be achieved. This includes management of processing resources as well as network resources both in end-systems and network nodes. In traditional router architectures, network resources are managed in a best-effort and rigid fashion. Programmable routers open the internal router details through object-oriented interface, enabling delivery of novel services as software packages by third-parties. These new services should be highly customizable. Network resources thus need to be controlled in a fine grained manner and could be bound with a service dynamically. A generic router resource interface [5] based on the programmable interfaces being proposed by IEEE P1520 [6] makes this possible by providing generic abstractions and dynamic binding capability. On the other hand, in an active network service modules share the resources in parallel in run-time, their access need to be synchronized to prevent conflict. To optimally utilize the limited resources in a router, more intelligent allocation of resources is preferred than fixed partition and reserva-

tion. This is particularly important for bandwidth, which nowadays becomes a commodity for auction.

The DPE also allows dynamic deployment of components which will be downloaded to an active node and run on the node's DPE. Security has to be considered on what a downloaded component is allowed to do as well as what resources it may consume. The dynamic deployment of components requires a special installation service as part of a computing model for active networks [2].

These form the basis for higher level reservation services. Because of the dynamic deployment those services are highly customizable by means of updating or exchanging components. With the definition of interfaces between instances of a service on different network nodes the service can provide a network wide resource control. Policies are used to regulate resource usage: user identities, time slots, priorities, etc. are used to gain efficient multiplexing of available physical resources.

The next sections describe these parts of the active DPE. The binding framework is used to set up communication paths to remote objects, the resource control framework is used to get a generic thus fine grained access to resources, the installation service allows the dynamic deployment of components, and finally a distributed reservation service is outlined to show how to take advantage of an active DPE to provide end-to-end QoS for distributed object applications.

2 Binding Framework

Communication between objects supported by the framework is through bindings, which are created by *object adapters*. 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 extends to the client side. The notion of ORB core in CORBA can be recovered as a specific, default object adapter that can be combined with other object adapters.

This flexibility allows 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.

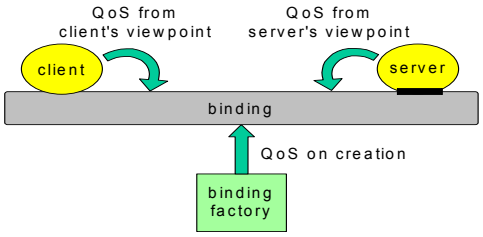

Fig. 1. QoS involved in object binding.

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 interface between the untyped world of protocols and the typed world of language bindings.

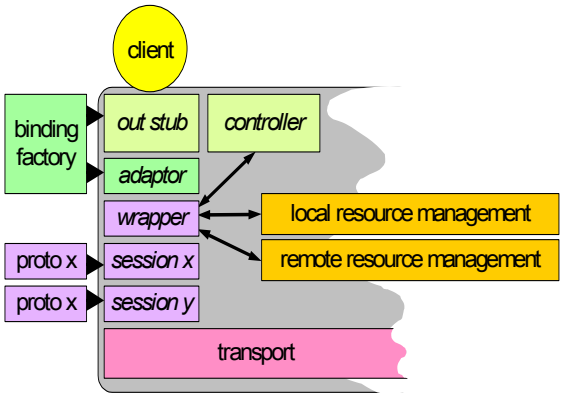

Fig. 2. Architecture of a binding (client side).

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

controller is introduced with the purpose to control the behavior of the binding and allow dynamic modifications of binding properties. This functionality can also be provided to the application layer.

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.

3 Resource Control Framework

The resource control framework provides a set of abstractions needed by system designers, service suppliers and application programmers to build applications requiring and/or providing QoS properties. These abstractions address fields of concern that must necessarily be considered when dealing with such QoS properties. Operating systems or platforms do not need to implement such abstractions but they must propose to the programmers basic services on top of which such abstractions can be built.

The first goal of this resource control framework is to provide basic abstractions for designing and engineering:

- resource multiplexing and scheduling mechanisms;
- QoS handling mechanisms.

The second goal of this framework is to provide guidelines for how to build "smart" resources and multiplexers for applications dealing with QoS constraints. The abstractions are therefore used to identify resource control design patterns.

In order to be effectively instantiated and to execute, objects must be mapped onto hardware resources such as memory, network, external data storage, processors etc. The mapping is done by *resource managers*. The role of a resource manager is to let a resource, or a set of resources, be shared between objects. A manager will provide to these objects an abstract view of the resources it manages, and control the way these resources are used. Resource managers have to keep track of what resources have been granted to which identities. This is important for logging and enables higher services like accounting. It is also crucial for ensuring that components cannot exceed predefined restrictions of resource usage.

The generic way of gaining access to resources is first to check the admittance to resources and then reserve them. If resources are not needed any more they get unreserved. The semantic of the admit/reserve pattern is that resources which have been admitted to a particular object stay so only for a predefined period of time. If the resources are not being reserved in this period the admittance will become invalid and later reservations may fail. The admit/reserve pattern allows to check the availability of a chain of resources before issuing the reservation. This is essentially important for end-to-end QoS. Nevertheless concrete implementations of schedulers may - for the sake of simplicity - choose to put the admit and reserve operations into one operation.

There are three major types of resource managers: *schedulers* manage the sharing of processing time, *memory managers* manage the sharing of memory resources, and *node resource managers* manage the sharing of local node resources. This paper focuses on node resource managers.

4 Node Resource Manager

A node resource manager (NRM) is seen as an active network facility to control the resources in a programmable router, e.g. bandwidth, queue, buffer, etc. It is deployed in each network node and responsible for managing the use of local node resources. It is the kernel module in an active distributed processing environment to support network-wide services with respect to resource access and usage.

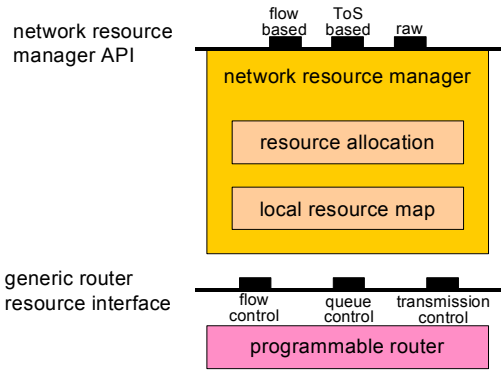

Fig. 3. Architecture of the node resource manager.

Figure 3 depicts the role of a NRM in the context of an active DPE. A NRM defines mechanisms that control the allocation of resources and synchronize the access. It makes use of the generic interface specified in [5], and provides a generic resource API to network-wide services, including high level reservation service.

In an active network, resource allocation represents one common request from network service modules, e.g. an admission control function. It could be a request for a minimum bandwidth for a flow, a class of service for packets, or a forwarding priority for a flow with particular protocol identifier. It is the major function that a resource manager should provide. To more flexibly support the resource needs from different services, and to maximize the resource utilization, a NRM implements adaptive allocation facility. The facility dynamically adjusts the allocated resources to accommodate new resource requests.

The necessity of such a facility can be justified by a simple scenario - in an intranet a flow from a director has higher priority than the flow from a normal employee in an active network. Data flows are thus supposed to have different priorities for transmission. This requires a more flexible/dynamic configuration of the limited network re-

sources, to optimally fulfill different users' resource requirements. That is, high priority flows should have precedence against lower priority flows when resources are allocated. As the overall resources are limited, a NRM should be able to dynamically re-allocate resources to accommodate new higher-priority flows, and become adaptive.

To allocate resources, a NRM maintains a view of the available node resources, mainly the bandwidth, and the state of the queues that split the overall bandwidth. It also maintains a view of the QoS-related parameters that a router is allowed to operate, e.g., discarding priority, queuing priority, and so on. These information together form a local resource map, which may be associated with allocation requests currently alive, to monitor the resource usage.

An allocation process generally consists of several basic steps: look-up, partition and admission. A look-up operation checks the available resources from the local resource map; a partition operation allocates required amount of resources; and the admission operation notifies the service module about the success or failure. On success, a soft-state is maintained for this allocation, and updated periodically so that an allocation can be modified later and use of the allocated resources can be monitored. Notably, the partition operation becomes more intelligent to realize adaptive allocation. In the following an example depicts the principle of a adaptive resource allocation and its result.

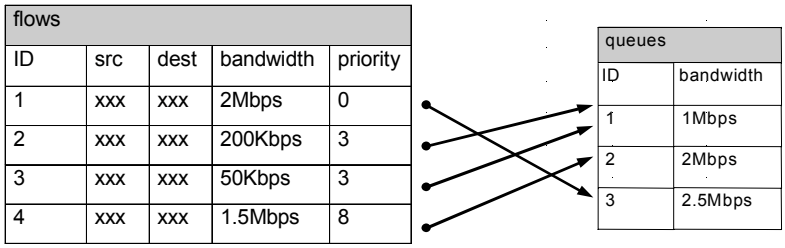

Fig. 4. Mapping of flows to queues.

Figure 4 shows the mapping of flows to queues. A local resource map maintains a table "queues" that tracks the state of all the queues, and a table "flows" to keep record of all the flows that request an amount of bandwidth. Each flow is associated with a queue where it obtains requested bandwidth. The rationale for adaptive allocation is preemption of resource occupation by flow priority, analogous to preemption of CPU time by thread priority. there are two key research issues to resolve in order to have a fair and efficient solution:

- Selection of lower-priority flows: grabbing resources from low-priority flows and allocating them to higher-priority requests also means violation of previous guarantee promise. Such a violation should be within the tolerance as defined in a service level agreement (SLA). Thus when selecting low-priority flows, a cross checking between flows' priority, their bandwidth and associated SLAs is required, and appropriate algorithms should be defined to be fair to each flow and its user.

- Re-allocation of resource: the resources allocated to a single low-priority flow might not be sufficient for a new flow with higher-priority, a merging of multiple flows' resources is preferable. In some other cases, resource partition is needed to accommodate more than one high-priority flows. An efficient scheme is to be researched to avoid waste of resources and operator-defined policy should be supported.

By this technology, we aim to realize the goal – efficient allocation and fair usage of network resources in an active network. A NRM should provide a generic resource manager API that a wide range of QoS services are able to use. Considering the major QoS frameworks, Intserv and Diffserv, we define an API that supports both flow-based and Type of Service (ToS)-based resource allocation.

Specifically, the flow-based API allows identification of flow, assignment of flow priority, required QoS. In addition, QoS tolerance and necessary notifications are also supported by the API. The ToS-based API allows identification packets with particular ToS value, mapping between ToS and QoS. In this API, a ToS value represents the transmission priority of a packet and can be rewritten and remapped to output priority by the NRM to achieve dynamic resource allocation.

5 Active Component Manager Service (ACMS)

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:

- Active components are installed via a policy-controlled way from internal or external repositories
- Policies are defined for the resource usage and allowed behavior of a component and the overall system. 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 agent) or could be transferred to the active node by other third entities. The ACMS 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.

5.1 Architecture

The architecture of the ACM is depicted in Figure 5. The main components are:

- *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 interacts with Policy and Credential Managers in order to take security decisions and grant or deny 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.
- *Policy Manager:* The policies for component/service access are maintained by this component. Via its interface authorized entities can dynamically modify the policies in the EEs or those of the node.
- *Credential Manager:* This component is responsible for managing the credentials of users/AC e.g. Certificates, public/private keys etc.

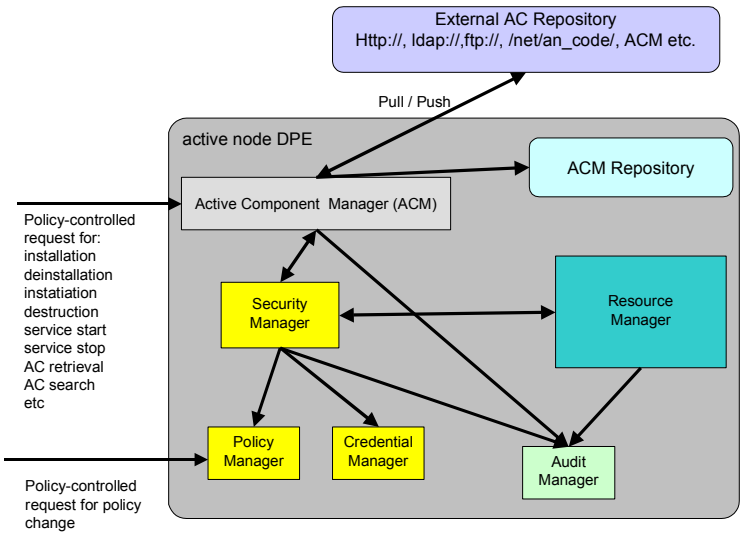

Fig. 5. Active Component Management Service Architecture

- *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 could be an external location accessed via known protocols such as http, ftp, ldap or even another ACM repository in another active node. Of course the AC could also reside somewhere in the local file system.

Figure 5. shows the procedure of installing an active component. A possible scenario that shows the interaction between the components 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 in co-operation with the Credential Manager (CM). Then it checks with the Policy Manager (PM) what the current policy is. The Resource Manager (RM) is consulted whether the action is allowed or not. 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, instatiation, 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 http, ldap, 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.

5.2 Implications of ACMS

Such a service has several implications in an AN infrastructure. We will try to comment here on the most obvious ones.

Security: The security of the AN is fortified as we can control via policy who installs what, where and for how long. Furthermore we can control who has permission at runtime to execute which components and under what environmental conditions (e.g. available memory). Also via the predefined node manipulation idea described later we can actually have an active node which is under the complete control of the node administrator and yet programmable by third parties.

Safety: The existence of the ACMS can guarantee a higher level of safety in the node. Many security violations occur not only because malicious software misuses the

node, but also from trusted code that does not execute correctly. So we need a way to be sure that the code that executes will not bring the node to an unstable state by mistake. To achieve this, one could use safe languages such as PLAN or Netscript, but usually this brings performance penalty and limits the programming flexibility. In our approach, run-time safety is also a task of the resource manager which monitors the resource usage of each component, and prevents access conflicts. ACM's role include setting-up a sandbox for each service which has its limited resource space, therefore providing a notion of safety at the instantiation stage. Furthermore the existence of ACMS allows the node owner to install his own AC on the node and allow third parties to call it and execute it. As he is the author of the code, he has already tested it and knows that this code is safe to use (something that is not the general case for code coming from third parties). Furthermore AN node programming is not considered a trivial activity and many programmers make different tradeoffs between code functionality and code testing. It is sure that the node owner will invest more effort in testing and debugging AC that he installs in order to avoid future problems, than the average user.

Predefined Node Manipulation: A lot of network operators are very much concerned with the idea of executing code within a node, mainly because of it obvious or hidden drawbacks such an action carries. For this category the ACMS can be a useful tool as it can provide specific interfaces to users to interact with the node. The network operator installs itself the necessary code and services in the node and allows the user to call this code with predefined and well tested parameters. Although again we have code executing we can predict the result of this execution since the node's status will change to one of the predefined ones. This can be seen as a hybrid approach since AC is executed (active network) but actually the node is manipulated via predefined interfaces (programmable network). This is a very attractive approach for network administrators that want to provide advanced functionality but are not willing to allow execution of foreign code into their nodes.

6 Distributed Reservation 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 [17] or immediate reservation (RSVP).

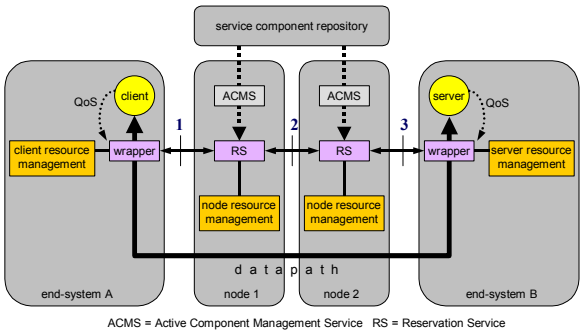

Fig. 6. Distributed reservation service for object communication.

Figure 6 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 intra-service interfaces (2) the several components can offer network-wide interfaces to end-systems (1 and 3). In the end-systems these interfaces are accessed out of the binding framework. A special wrapping session 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 on the active node by the ACMS. The run-time instance of a reservation service component has its limited resource space, allocated by the ACMS 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 naming service, 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 or not 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.

7 Implementation

A prototype of the active DPE is implemented in JAVA using the modular and extensible Jonathan ORB [14] as the basis for the binding and resource control frameworks. The active DPE is deployed in a testbed consisting of three Hitachi Gigabit Routers 2000 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 is 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.

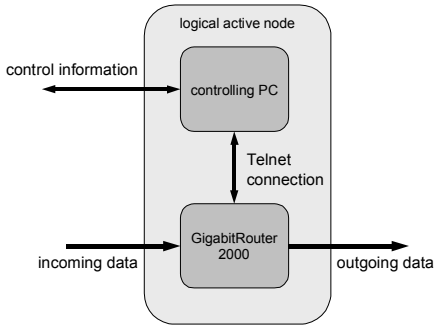

Fig. 7. Details of the logical active node.

8 Conclusion

The paper describes an ORB-based Active Networking approach. A framework for an active DPE is presented to integrate programmable networks, active networks, and distributed object technology. The framework supports the execution of network service as downloaded active components to provide QoS. These services flexibly program network resources through more dynamic and efficient resource manager interface. Their parallel execution is controlled, particular with respect to resource access and usage, to ensure safety. Distributed object applications obtain their QoS expectation with support of a generic reservation service, which is de-coupled from the under-

lying protocol and can be dynamically customized. In this framework, QoS support is transparently embedded in the communication stack as part of binding action.

Policy is generally considered important as a mean by network administrators to control the active network. A dedicated policy-based management system interacts with other major services such as resource control, installation, and reservation. A policy defines identities so that decisions can be made whether a request should be granted or denied. The role of policies is to associate identities with rules that determine the access and usage of resources.

As for any other framework the design of components is crucial. Plenty of care has to be taken when specifying the components' interfaces: new services or applications should be able to treat the offered components as building blocks and take advantage by composing the provided functionality.

References

1. Aurrecoechea, C., Campbell, A.T. and L. Hauw, "A Survey of QoS Architectures", ACM/Springer Verlag Multimedia Systems Journal , Special Issue on QoS Architecture, Vol. 6 No. 3, pg. 138-151, May 1998
2. D. S. Alexander, "ALIEN: A generalized computing model of active networks", dissertation in computer and information science, University of Pennsylvania, 1998
3. Ted Faber, Bob Braden, Bob Lindell, Jeff Kann, Graham Phillips, Alberto Cerpa, Active Nets Workshop, April 1999.
4. Hitachi and GMD-FOKUS co-operation project.
   URL: http://www.fokus.gmd.de/research/cc/glone/projects/bang/
5. Deliverable 7 as a technical report for Hitachi-FOKUS BANG project.
6. Jit Biswas, et al., “The IEEE P1520 Standards Initiative For Programmable Network Interface”, IEEE Communications Magazine, Oct. 1998.
7. Campbell A.T., "QOS Architectures" ,(Eds.) M. Tatipamula and B. Khasnabish, Multimedia Communications Networks , Artech House Publishers, Chapter 3. pg. 103-128, ISBN 0-89006-936-0, June 1998.
8. Object Management Group, "The Common Object Request Broker: Architecture and Specification", Minor revision 2.3.1, October 1999
9. Bruno Dumant, François Horn, Frédéric Dang Tran, Jean-Bernard Stefani, "Jonathan: an Open Distributed Processing Environment in Java", March 17, 1998
10. IETF RFC 2475, "An Architecture for Differentiated Services", Dec. 1998.
11. Frédéric Dang Tran, Jean-Bernard Stefani, "Towards an extensible and modular ORB framework", April 1997
12. Lazar, A.A., "Programming Telecommunication Networks", IEEE Network, September/October 1997, pp. 818.
13. ITU-T | ISO/IEC Recommendation X.901 | International Standard 10746-1, "ODP Reference Model: Overview", January 1995
14. Jonathan ORB URL: http://www.objectweb.org/jonathan/
15. IETF RFC 1633, “Integrated Services in the Internet Architecture: An Overview”, June, 1994. IETF Integrated Service Working Group.
16. IETF RFC 2205, “Resource Reservation Protocol (RSVP) – Version 1 Functional Specification”, September, 1997.

17. A. Schill, F. Breiter, S. Kühn, "Design and Evaluation of an Advance Reservation Protocol on top of RSVP", IFIP 4th International Conference on Broadband Communications, Stuttgart, 1998, Chapman & Hall, pp. 23-40
18. D. L. Tennenhouse, "A Survey of Active Network Research", IEEE Communications Magazine, 35 (1), January 1997, pp. 80-86.