Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture

Aurel A. Lazar, *Fellow, IEEE*, Koon-Seng Lim, and Franco Marconcini

***Abstract*—A conceptual framework, called the binding model, for the creation, deployment and management of multimedia services on ATM-based broadband networks with end-to-end quality-of-service (QoS) guarantees is presented. The key function of the associated binding architecture is to provide an open programmable environment that facilitates the easy creation of flexible services. We describe the implementation of a prototype binding architecture called *x*bind as a middleware toolkit for building open programmable ATM networks. Finally, we present our initial experiences with experimenting and deploying *x*bind over an ATM testbed and highlight some of the lessons learned.**

I. Introduction

We describe a conceptual framework (called the binding model [27]) for the creation, deployment and management of multimedia services on ATM-based broadband networks with end-to-end quality-of-service (QoS) guarantees. The model allows the binding of networking resources with the goal of creating distributed services across heterogeneous networking platforms. We present a service creation methodology and show how scalable multimedia distributed services can be constructed from a set of simple network services.

A key difficulty in network programmability is finding an agreeable definition of a service that exhibits operational significance. For example, the TINA-C [3] service architecture [42] defines a service as "...a meaningful set of capabilities provided by an existing or intended network to all who utilize it, like customers, end users, network providers, and service providers" whereas ANSA [8] defines it as the functional role of a computational object. In our architecture, we restrict our definition of services to a set of reusable capabilities for supporting the scalable development and deployment of multimedia applications with QoS guarantees. Thus, our notion of service encompasses high level activities related to multimedia distribution like video on demand as well as the low level support functions needed to realize such a service.

Based on this understanding, we observe that typical multimedia distribution services require support from four dimensions. Specifically, these services need support for name and resource mapping, support for resource management and reservation, support for a media stream transport and support for service management functionality. They require name and resource mapping in order to translate the logical service abstractions into physical resources. They also require a resource reservation procedure since the service must guarantee QoS, they require a stream transport for distribution of its media streams with QoS and finally they require management functionality for monitoring and control. It is also implicit from the description above that some form of state abstraction is needed for characterizing the dynamic elements of the service while it is executing.

Manuscript received May 7, 1995; revised March 15, 1996.

The authors are with the Department of Electrical Engineering and Center for Telecommunications Research, Columbia University, NY 10027-6699 USA.

Publisher Item Identifier S 0733-8716(96)06982-X.

The binding architecture [24] is a formal description of the binding model. Its principal aim is to provide an open programming environment that facilitates the easy creation of distributed services. By open, we mean the architecture must support functional application programming interfaces (API's) for resource control and management that service providers can use for developing useful services. By programmable, we mean that these API's should be "high-level" enough to allow the service specification and creation process to be carried out via a high-level programming language.

The Binding Architecture consists of an organized collection of interfaces, called the binding interface base [28] (BIB), and a set of algorithms that run on top of these. BIB interfaces provide an open and uniform access to abstractions that model the "local" states of networking resources. Binding algorithms play a key role in the service creation process through the process of interconnecting (binding) networking resources. QoS is explicitly modeled in the architecture via a set of abstractions that characterize the multiplexing capacity of networking and multimedia resources under QoS requirements. These abstractions allow admission control tests to be executed in real time during the resource reservation process.

We describe our initial experience with an implementation of the binding architecture, called **xbind** [39], a middleware toolkit for building scalable multimedia services on top of heterogeneous computing and networking platforms. **xbind** achieves interoperability between the different resource control and management algorithms of a multimedia network by giving these open access to the BIB. By adopting the common object request broker architecture (CORBA) [37], an industry standard for distributed computing platforms, **xbind** achieves interoperability in terms of data representation across diverse operating systems and machine architectures.

This paper is organized as follows. In Section II, the binding model as a framework for network programmability is introduced. To give the proper context, an extended reference model (XRM) [22] for networking is first introduced. The Binding Model is identified as a service creation model within one of the components of the XRM. Finally, the service creation model is briefly described. In Section III, elements of

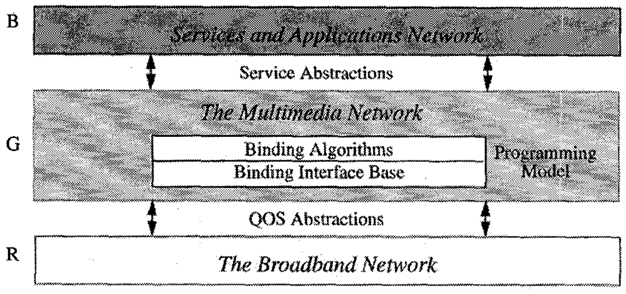


Fig. 1. Overview of the RGB decomposition of the XRM.

the binding architecture are given. These include a description of the BIB and a specification of the service creation process. In Section IV, our experience with **xbind** is discussed. Related work is presented in Section V. Finally, concluding remarks appear in Section VI.

II. The Binding Model as a Framework for Network Programmability

The ability to create multimedia services requires three key components. First, there must be a means of obtaining information about the state of resources in the system. Second, a middleware layer is necessary for providing appropriate state abstractions. Third, a general service architecture is needed for structuring the relation between services and their interactions so that more complex services can be systematically composed from simpler ones. To put the latter in the proper context, we will start our presentation by giving more structure to the XRM. This will be followed by a description of the Binding Model and its embedding into the XRM. We will close this section by discussing the associated service creation model.

A. The RGB Decomposition of the XRM

The XRM models the communications architecture of networking and multimedia computing platforms. It consists of three components called, the broadband network, the multimedia network, and the services and applications network (see Fig. 1). The broadband network is defined as the physical network that consists of switching and communication equipment and multimedia end-devices. Upon this physical infrastructure, resides the multimedia network whose primary function is to provide the middleware support needed to realize end-to-end QoS guarantees over the physical media-unaware network. This is achieved by abstracting from the broadband network a set of QoS abstractions. Based on the latter, resource management and control can be performed. However, QoS abstractions, by themselves, are a passive representation of resource states. Services, on the other hand require activity in terms of resource reservations and distributed state manipulation. These activities can be viewed as part of an algorithm which when executed creates the service. In this perspective, the multimedia network provides a programming model which together with the QoS abstractions it receives from the broadband network, allows service behavior to be specified and executed. Service abstractions represent the states of a service created using algorithms native to the multimedia network. These abstractions are used by the services and applications network for managing and creating new services through dynamic composition and binding.

The three component networks of the model above can be refined further. As shown in Fig. 2, each of the original RGB component networks is decomposed into five planes. The decomposition results in three submodels are collectively known as the RGB decomposition.

The RGB decomposition represents detailed viewpoints of the broadband network, the multimedia network, and the services and applications network, respectively. The interface between R-, and G-models is a set of QoS abstractions typically structured as graphs that quantitatively represent various resources in the physical network. The G-model uses these graphs for creating service abstractions that are provided to the B-model for building more complex services. Thus, the interface between the R- and G-models and the interface between the G- and B-models are abstractions that are similar in structure but differ in usage. In the following paragraphs, we describe each plane of the RGB models in detail. Throughout this document the full scoping notation will be used to refer to a particular plane of the XRM (e.g., XRM::M) or a submodel (e.g., XRM::G). The interface between planes will be denoted by using two bars. For example R||G represents the interface between the R- and G-models.

The N-plane functionality of the R-model is one of network and system management and comprises monitoring and control of individual states. These states might correspond to the status of a link, the temperature of a interface card, etc., and are enveloped as managed objects residing in the MIB. A client/server interaction is the basis of the N-plane manager/agent model for monitoring and controlling network elements. The M-plane models the resource control tasks. For example, at the switch or multiplexer level the main resource functionality is in terms of buffer management and link scheduling [13]. At the workstation or PC level these same tasks appear as operating system scheduling and memory management. Flow control is another important resource control mechanism—it acts on the frame/cell level. The D-plane abstracts the main network components, i.e., switches, multiplexers and media processors as a global distributed memory. Specifically, communication links are modeled as FIFO memory and, switches and processors as random access memory. These and higher level abstractions thereof are entities included in the management information base (MIB). The C-plane supports exchange of state information among distributed buffer management and link scheduling entities. Exchange of state information is required in cooperative distributed scheduling and buffer management [26]. Finally, the U-plane of the R-model defines cell level adaptation protocols for segmentation and reassembly as well as reliability checks. This includes the ATM layer and the ATM adaptation layer functions.

Monitoring the behavior of distributed systems is a key requirement for the N-plane of the G-model. The task of the N-plane is, among others, the monitoring of distributed object oriented systems (such as CORBA) and their interactions. The M-plane offers orchestration as well as other resource

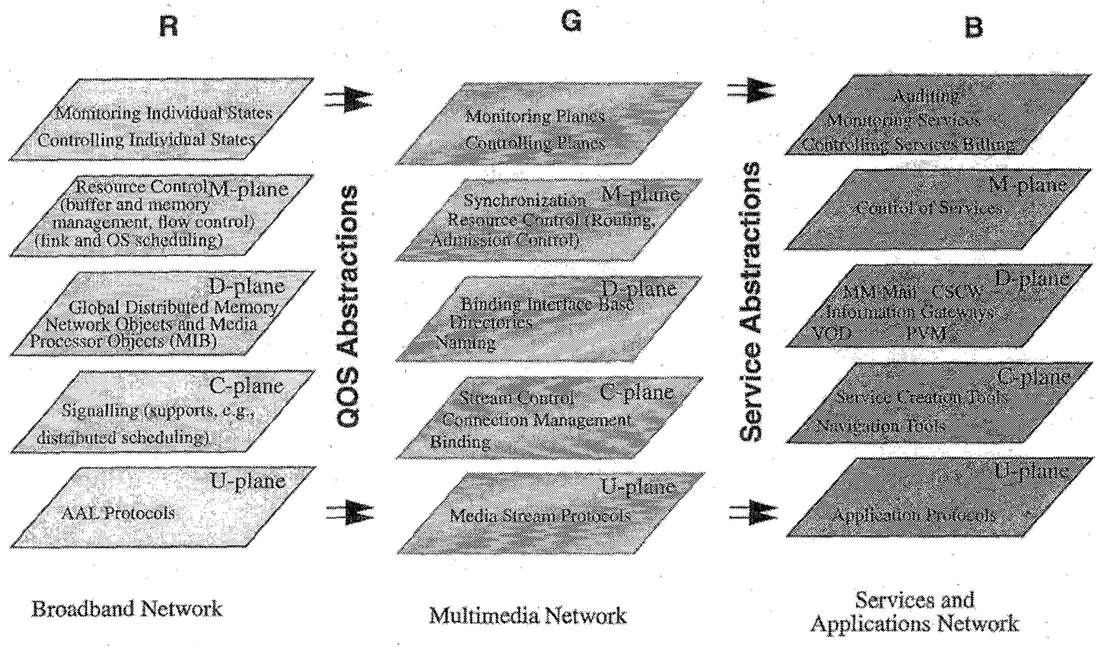


Fig. 2. The RGB decomposition of the XRM.

allocation mechanisms. The key resource allocation algorithms are routing and admission control. The D-plane consists of a BIB, a distributed repository containing information about entities that might participate in the creation of network services (a binding process). Services are defined as a set of interconnected resources. In addition, there is a need for directory, trader or broker, and naming services. The C-plane supports stream control as well as connection management and, more generally, distributed algorithms. Stream control refers to protocols required for remote control of multimedia devices such as tape drives, multimedia on demand systems, etc. Both unicast as well as multicast connection management algorithms belong to the C-plane. Finally, the functionality of the U-plane of the G-model includes the support of a number of media stream protocols such as a native ATM stack [12] and other real-time protocols. These protocols can co-exist with a number of already widely used transport protocols that offer best effort service (such as TCP).

The management of services is a functionality of the N-plane of the B-model. Here we identify management support for access, security, configuration, billing and auditing services, among others. Service admission control is defined both in the N- and in the M-plane. How to control services and negotiate networking and computational resources is a key requirement of the M-plane. The D-plane is a repository of services such as multimedia mail, computer supported cooperative work, video on demand, parallel virtual machines, etc. Navigation and service creation tools are the services of the C-plane. Finally, application protocols belong to the functionality of the U-plane.

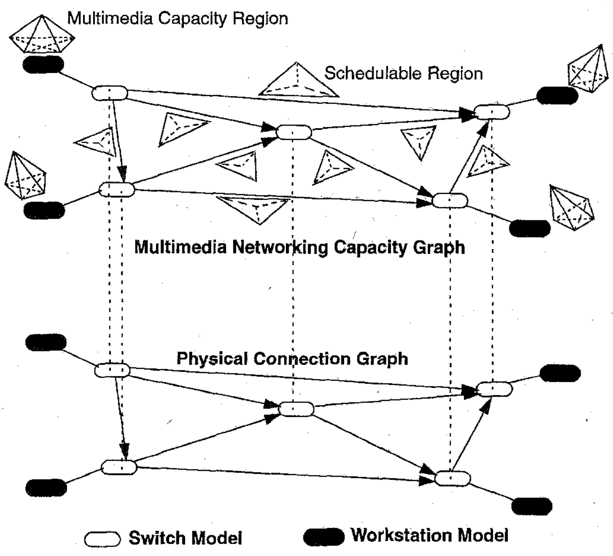

Multimedia Capacity Region
Schedulable Region
Multimedia Networking Capacity Graph
Physical Connection Graph
Switch Model
Workstation Model

Fig. 3. The physical connection graph and multimedia networking capacity graph.

B. The Binding Model

The binding model defines a conceptual framework for the creation, deployment, and management of multimedia services on ATM-based broadband networks with end-to-end QoS guarantees. Binding refers to the activity of creating a requested multimedia service with QoS guarantees. It entails the association of a set of network and end system resources with a media transport protocol and the service management system.

The binding model gives the XRM::G its operational capability through a model of service creation. In this context, service creation is defined as a process of object composition whereby objects are manipulated by algorithms residing in various XRM::G-planes. The binding model describes how to marshall and compose resources into network services starting with services offered at the R||G interface.

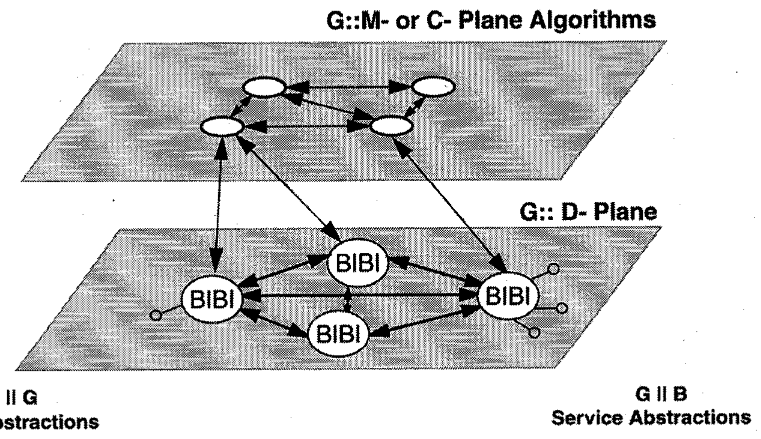


Fig. 4. Network view of the service creation process.

There are a number of services provided at the R||G interface including the physical connection graph [22] and the multimedia networking capacity graph (Fig. 3). The physical connection graph carries all the topological information about the physical network and the associated resources. Fig. 3 also depicts the multimedia networking capacity graph. The capacity of networking as well as media resources is characterized through the concepts of schedulable and multimedia capacity regions as given in [25]. Suffices to say here, that the multimedia networking capacity graph consists of a set of interconnected interfaces denoting the access to objects representing quality of service abstractions. There are also a number of other services offered at the same interface such as network management services in support of plane management. However, because of space limitations, these will not be covered here.

The binding model consists of two building blocks: a set of (enveloped) states organized in the BIB and a set of binding algorithms operating on these interfaces. The two building blocks structure of the binding model is justified through a separation principle. This principle gives a clear focus toward what should be and what needs to be standardized within the XRM::G. Our proposal is to standardize the access to the BIB and leave the algorithms and the service creation proprietary.

The BIB is a repository of interfaces modeling the access to resources such as switches, links, multimedia devices, etc. Within the XRM, the BIB is located in the Telebase (G::D-plane). The physical connection graph and the multimedia capacity graphs are both logically represented in the BIB.

At the G||B interface, the XRM::G model also offers a number of network services. Among others, we mention, virtual circuits, virtual paths [1], virtual networks [2], and multicast [5]. The reader is recommended to consult these references for details.

C. A Framework for Service Creation in the Binding Model

The functionality of the G-Model is one of mapping the services provided by the R-Model into network services provided to the B-Model at the G||B interface. The network view of service creation describes how binding algorithms that participate in the process of service creation interact and co-exist in the G-Model.

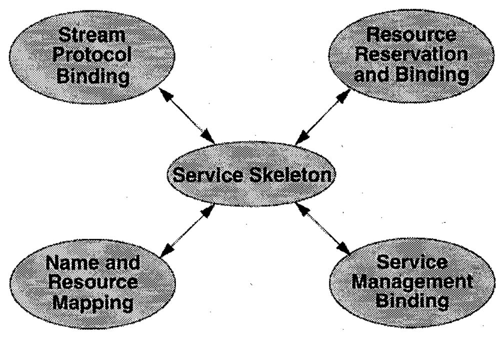

Stream Protocol Binding
Resource Reservation and Binding
Service Skeleton
Name and Resource Mapping
Service Management Binding

Fig. 5. Service view of the service creation process.

The process of service creation consists of generating, starting from a set of objects (states) residing in the BIB, another set of objects (states). The generation process is executed by a service provisioning entity. Multiple such entities execute in parallel and in a distributed fashion. The role of the Binding Model is to define the overall organization of distributed operations for service creation. These operations or services can be used to create other services. All operations reside in the N-, M-, C-, and U-planes.

Fig. 4 shows the distributed nature of BIB interactions, and suggests possible distributed interactions among different binding algorithms during the service creation process. Binding algorithms arise in connection set up for broadband networks, distributed systems implementing synchronization protocols, resource allocation protocols such as routing, multimedia computing platforms, etc. New applications can be supported without changing the underlying binding model. In addition, several proprietary binding algorithms can operate at the same time.

The service (or local) view toward the service creation process highlights the steps that lead to the creation of the service. The process of service creation consists of five steps.

1) Create a service skeleton for an application such as a virtual circuit, virtual path, virtual network or multicast. The structure of the skeleton for a virtual circuit, for example, consists of a graph from a source node to destination.

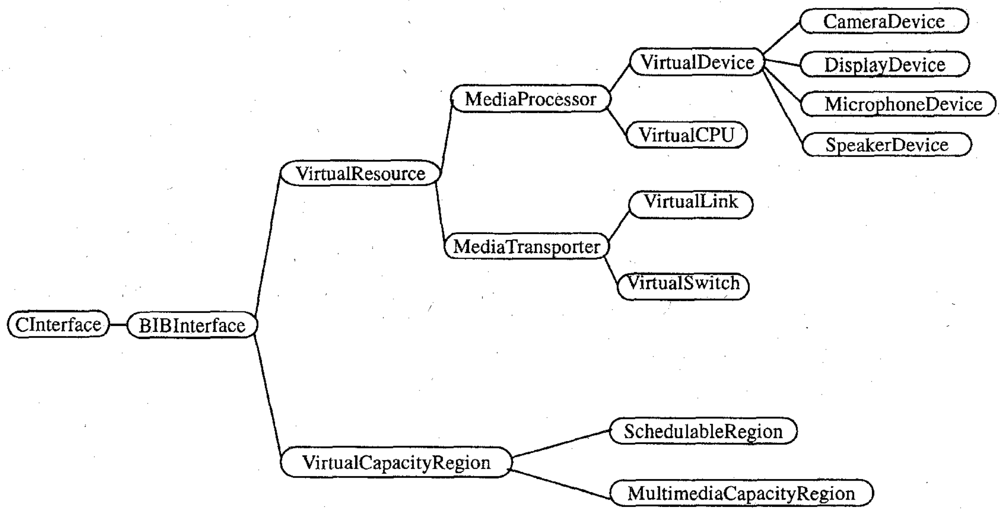


Fig. 6. The binding interface base.

2) Map the skeleton into the appropriate name and resource space and thereby create a network application.
3) Associate (bind) to the application a media transport (stream) protocol and thereby create a transport application.
4) Bind the transport application to resources and thereby create a network service.
5) Bind the service management system to the network service and thereby create a managed service.

Fig. 5 illustrates the individual service view of the service creation process. First, a skeleton is created using name and resource mapping services [5]. The resulting network application is bound to a transport protocol, resources, and service management. When a service is required, the application process issues a service request to the responsible service provisioning entity that in response will invoke the corresponding binding algorithm(s) for establishing a service instance.

III. Elements of the Binding Architecture

The binding architecture is a formal description of the binding model. In what follows, we will presents some of its better understood elements. These include the BIB, a switch control API, a set of broadband kernel services, and a model for developing high level broadband services from these elements.

A. The Binding Interface Base

The binding architecture specifies all interfaces defined in the BIB in CORBA's interface definition language (IDL). Because the focus of the BIB is on abstracting only resource states, a number of interfaces present in the original proposed BIB of [23] have been omitted. The new streamlined binding architecture BIB is given in Fig. 6.

Besides interfaces, the BIB also defines a specialized class of factory interfaces for instantiating all other interfaces. These interfaces are used during the bootstrap of the system for creating the appropriate interfaces to existing physical and logical devices. The IDL definition of the generic BIB factory interface called the *BIBFactory* allows the creation, destruction of an interface and querying of the list of interfaces currently instantiated by a server. All other factory interfaces are derived from this interface and customized to create specialized interfaces specific to the resource concerned.

The *VirtualSwitch* controls the manner in which VPI/VCI pairs are allocated and deallocated. The VirtualSwitch interface gives therefore access to the abstraction of the valid VPI/VCI pairs for both an ATM switch as well as an ATM adapter card. Obviously, the two have substantial hardware differences: in the case of the switch the methods of the interface are mapped to the functionality of the hardware. In the case of the adapter, however, the fabric component is absent. We expect that in the future the internal bus of the workstations or PC's will have a design very close to an ATM switch and thus VC's will terminate not onto an adapter but directly to the target or source device. The primary methods of the VirtualSwitch interface are for requesting for free VPI/VCI pairs as well as for specifying the mapping between incoming and outgoing VPI/VCI pairs in a switch fabric. These are given by the methods getOutputChannelIdentifier(), getInputChannelIdentifier(), and the set of commitChannel() calls, respectively (see Table I).

On the other hand, the *VirtualDevice* interface abstracts the functionality of a multimedia stream device. The current interface models a multimedia device as either a source or a sink. The interface allows multimedia streams to be added to a device or dropped from a device by specifying the end VPI/VCI's used by the stream. Individual active streams can also be paused or resumed. A list of these methods is given in Table II.

B. The Switch Control API

The implementation of the hardware specializations of the VirtualSwitch for all the platforms listed in Table III required considerable development effort. Our goal of having access

TABLE I
VIRTUALSWITCH INTERFACE

```
typedef short VCI;
typedef short VPI;
typedef short PortId;

interface VirtualSwitch: MediaTransporter {
                                   . . . . . . . . . . . . . .
   BIBStatus setInputChannelIdentifier(in VCI invci, in VPI invpi,
      in PortId inpid);
   BIBStatus setOutputChannelIdentifier(in VCI outvci, in VPI outvpi,
      in PortId outpid);

   BIBStatus getInputChannelIdentifier(inout VCI invci, inout VPI invpi,
      in PortId inpid);
   BIBStatus getOutputChannelIdentifier(inout VCI outvci, inout VPI outvpi,
      in PortId outpid);

   BIBStatus commitChannel(in VCI invci, in VPI invpi, in PortId inpid,
      in VCI outvci, in VPI outvpi, in PortId outpid);
   BIBStatus removeChannel(in VCI invci, in VPI invpi, in PortId inpid,
      in VCI outvci, in VPI outvpi, in PortId outpid);

   BIBStatus commitInputChannel(in VCI invci, in VPI invpi);
   BIBStatus removeInputChannel(in VCI invci, in VPI invpi);

   BIBStatus commitOutputChannel(in VCI outvci, in VPI outvpi, in PortId outpid);
   BIBStatus removeOutputChannel(in VCI outvci, in VPI outvpi, in PortId outpid);
};
```

TABLE II
VIRTUAL/DEVICE INTERFACE

```
struct StreamInfo {
   short Id;
   VCI vci;
   VPI vpi;
};

typedef sequence<StreamInfo> streamList;

interface VirtualDevice: MediaProcessor {
   BIBStatus addStream(out short Id, in short dir, in VCI vci, in VPI vpi);
   BIBStatus removeStream(in short Id);
   BIBStatus getStreams(in short dir, out streamList idList);

   BIBStatus pauseStream(in short Id);
   BIBStatus resumeStream(in short Id);
};
```

to low level control API's that directly operate on the routing tables of various ATM switches is currently difficult to achieve since these are not open to third parties. However, through direct collaboration with the switch manufactures we were able to implement the VirtualSwitch on an individual basis. This approach is rather time consuming and, obviously, not scalable.

In order to speed up the implementation process, we proposed a set of API's, called the switch control application programming interface (SCAPI) [29], for the general control and management of ATM switches. The same motivation (albeit through a different approach) has also prompted Ipsilon Networks to propose a wired protocol called the general switch management protocol (GSMP) [36] with similar functionality to allow remote switch management. The main differences between the two proposals lie in the area of QoS control and their intended use. In the SCAPI proposal, QoS issues play a central role. In the current version of GSMP, however, QoS issues have not been explicitly addressed. Moreover while GSMP is meant to allow remote access to switch control functionality, SCAPI is a programming interface for developing high level control software on a switch. In the

TABLE III
NETWORK AND MULTIMEDIA DEVICES CURRENTLY SUPPORTED BY **xbind**

Binding Interface	Hardware Resource	Software Platform
VirtualSwitch		
Switches	Fore ASX-100 Fore ASX-200 NEC Model 5[1]	SunOS 4.1.3 SunOS 4.1.3 SunOS 4.1.3
Adapters	Fore Adapter 200-Series	SunOS 4.1.3, Solaris 2.3, 2.4, 2.5 UP-UX A.09.07/A.09.05 Irix 5.3
VirtualDevice		
SpeakerDevice MicrophoneDevice	Native Audio Devices of: • Sun Sparc 5,10, 20 • HP 9000 • SGI Iris Indigo	SunOS 4.1.3, Solaris 2.3, 2.4, 2.5 UP-UX A.09.07/A.09.05 Irix 5.3
CameraDevice DisplayDevice	Parallax Graphics Xvideo on: • Sun Sparc • HP 9000 SunVideo	SunOS 4.1.3, Solaris 2.3, 2.4, 2.5 UP-UX A.09.07/A.09.05 Solaris 2.3

1. The controller board of the NEC Model 5 switch was replaced with a bus converter attached to a Sun Sparc 2 workstation acting as controller.

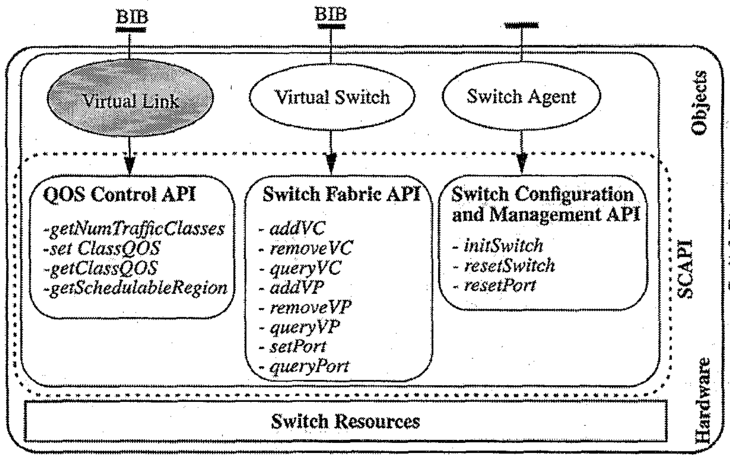


Fig. 7. Relationship between BIB interfaces and SCAPI.

Binding Architecture, this is represented by the software that implements the VirtualSwitch interface. Fig. 7 illustrates the SCAPI proposal.

C. Broadband Kernel Services

The broadband kernel services refer to low level "operating system" like functionality provided by the middleware for deriving higher level multimedia services. These include services for connection management, routing, device management, and transport. The connection management service provides connection setup and tear down functionality between a number of network endpoints. The routing service provides a route location functionality for traversing any two network endpoints. Together, they implement the basic connectivity services in the network component of the system and can be used to built switched virtual circuits (SVC's). On the host end, a device management service provides the functionality of tracking and managing the numerous multimedia devices spread throughout the system. Ensuring format compatibility and presentation translation are also aspects of this service. Finally the transport management service defines the functionality required for sustaining transport streams in the network. These include functionality for rate control and bandwidth renegotiation.

These services are realized in accordance to the service framework of the binding model. All services with the exception of the routing service (which currently implements only static shortest path) are encapsulated in servers which expose the four basic interfaces for service instance creation,

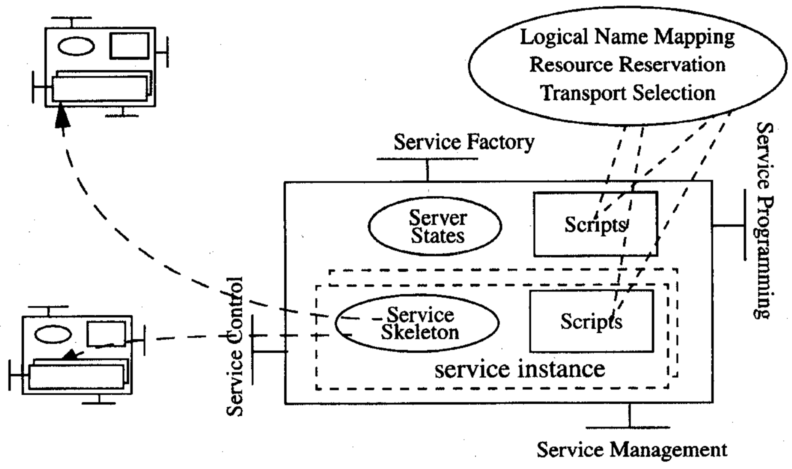


Fig. 8. Interfaces for multimedia distribution services.

programmability, control and management. The relation and the interplay between these services and their interfaces will be detailed in the next section.

D. Broadband Services

Based on the service creation process described in Section II-C, we define the following object framework for service creation [30]. Services are offered by servers which support the creation and maintenance of their state. When a request for a service is made to a server, a service instance is instantiated or created. Each service instance is composed of an algorithmic part and a data part. The algorithmic part expresses the logic of the service instance while the data portion is an abstraction of its state. Individual service instances can be customized by modifying the logic of its algorithmic part. Similarly control can be effected on an executing service instance by the modification of its states. Servers also have an algorithmic component which specifies how service instances are created, deployed and managed and a data part which models the state of the server and its policies.

Typical servers expose four interfaces. These are the service factory interface, the service programming interface, the service control interface, and the service management interface. The service factory interface is used to request the creation of a service instance. The service programming interface allows customization or modifications to be made to the algorithmic component of the server or service instance. The service control interface is the operational interface to the service instance and allows the monitoring and manipulation of service instance states during execution. Finally the service management interface allows for monitoring and control of the server and the setting of management policies as Fig. 8 below illustrates.

It is possible for some services to have no state (e.g., a simple database lookup service). Such services do not create instances and have only factory and logic interfaces. We exemplify the operational significance of the service creation process by briefly describing below a realization of a video conferencing service.

As an example, consider a simple video conferencing service as illustrated in Fig. 9. In order to realize this service, a small number of supporting services must be defined. In this case, we say that the video conferencing service is composed of a transport management service, a device management service, a connection management service and a route management service. The video conferencing session manager upon receiving a request for a new session from its service factory interface activates two other services, the connection management service (point 1) and the device management service (point 2). The connection management service contacts the routing service (point 1.2) to obtain a source route and then proceeds to set up the connection (point 1.3). The device management service first checks to see if the end-devices involved in the session are compatible and have correct access permissions before activating them (point 2.1). The information passed between the device manager and the devices include the VCI/VPI's to use as well as the media stream protocol for transport. The end-devices upon activation register themselves with the transport manager (point 2.2) which in return registers itself with the video conference manager (point B). If during the course of the session the end-devices detect congestion or fault in the network, they can trigger transport renegotiation through the control interface of the transport manager (point A). The transport manager in turn can trigger the control interface of the video conference manager to request corrective action through its control interface (point B). Similarly if there is a user request for change of bandwidth or QoS requiring transport action, the video conference manager can trigger the control interface of the transport manager (point C) which in return can trigger corrective flow control actions on the device (point D). A service manager can monitor and manage the video conferencing service through its management interface (point E) or modify the service logic (e.g., to upgrade or enhance the service) through its logic interface (point F).

IV. EXPERIMENTING WITH **xbind**

Since the Fall of 1994, we have been experimenting with **xbind** [39], an implementation of the binding architecture. The

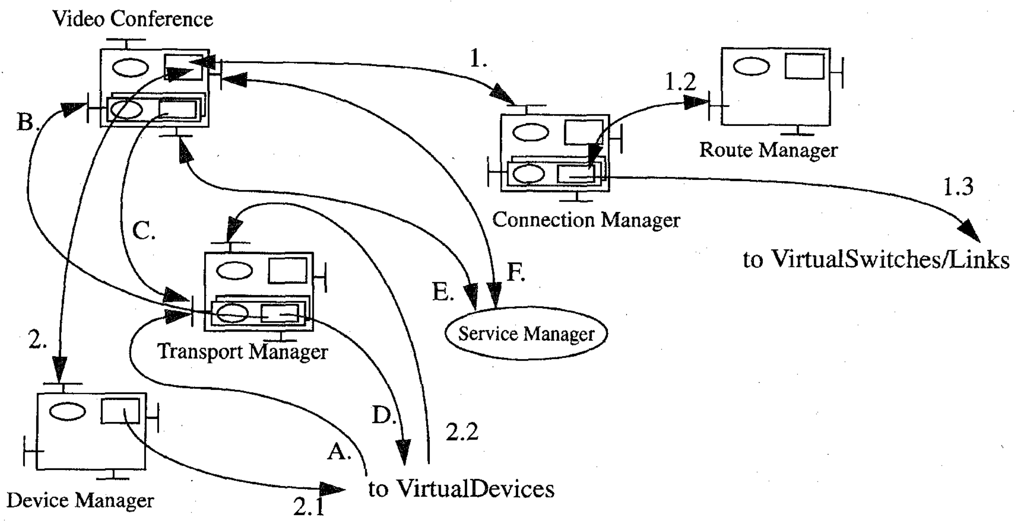


Fig. 9. High level video conferencing service architecture.

objective of **xbind** is the system level validation of the binding model. **xbind** currently supports a subset of the functionality defined in the binding architecture. Important components whose implementation has been left to future releases are the mechanisms for supporting QoS for some of the resources. We delayed the implementation of this important part of the Binding Architecture because of the lack of support in the first generation ATM switches for traffic classes with associated QoS specifications. Similarly the currently available workstations or personal computers cannot guarantee allocation of computational resources, and therefore any implementation of the multimedia capacity region mentioned in [23] would have been very limited in scope. We decided instead to focus the first implementation of **xbind** on interoperability aspects.

In the following sections, we will examine some important elements of **xbind** including its signalling infrastructure, its deployment, and from an implementational perspective, an example service, the video conferencing service already described in Section III-D.

A. The Basic Signalling Infrastructure

The term "signalling" refers to the functional role of the *C*-plane, namely the transport of control information. From this perspective, the basic signalling infrastructure is somewhat akin to the CCSS #7 of the telephone network or the distributed processing environment (DPE) of the TINA reference model. In **xbind**, the signalling infrastructure is composed of a collection of CORBA object request brokers (ORB's) which provide a homogenous name space across a distributed set of heterogeneous platforms. The signalling infrastructure also provides location transparency and well defined call semantics through its remote procedure call (RPC) interface.

Using CORBA for signalling requires having CORBA objects reside directly on the networking or media resources—it requires the capability to run CORBA objects such as the VirtualSwitch on the ATM switch hardware. In our case where an off-the-shelf CORBA implementation by Iona Technologies (Orbix [13]) was the platform of choice, this was possible for some of the earlier generation groupware ATM switches (like the Fore System ASX-100 and ASX-200) which employed a conventional microprocessor with standard UNIX operating system as the switch controller. On newer switch platforms where the controller runs on an embedded board with a real time operating system, realizing a similar configuration would be more difficult.

One of the drawbacks of the current generation of CORBA implementations is that these are usually not supported by different lower level transport protocol stacks. As a result, there is usually little alternative to the use of TCP/IP. We did some preliminary investigation with Postmodern Technologies on the possibility of replacing IP with a native ATM network protocol stack in their ORBline product but concluded that it would require too much effort in rewriting portions of the ORB.

In addition, this aspect of CORBA implementations turned out to be secondary to the initial interoperability objective of the Binding Architecture. Moreover, the typical semantics of object invocations favored short messages on a highly flexible connectionless transport, which in fact made IP the ideal transport! We therefore concentrated on solving the problem of how to support IP [21] communication services on an ATM network prior to the establishment of any wired signalling protocol.

To this end, we decided on a combination of two different approaches to solve the problem, namely:

1) Whenever possible, we took advantage of a switch's native signalling mechanism for establishing IP over ATM. In the case of Fore System switches, the switches automatically create SVC's for the IP traffic using SPANS during bootup.
2) Otherwise, an IP network is configured through scripts that add static IP routes or create *ad hoc* VC routes in the routing table of the switches through CORBA calls to the local VirtualSwitch interfaces.

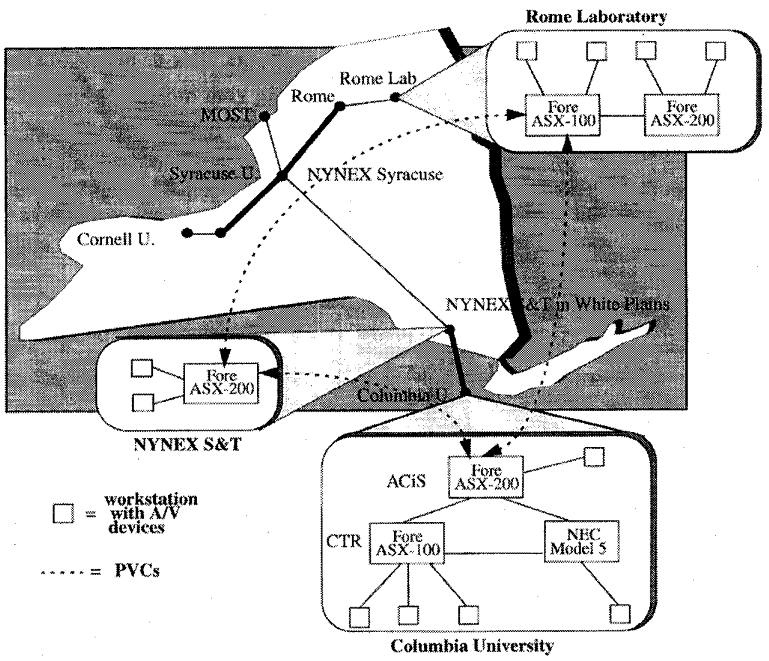


Fig. 10. Deployment of **xbind** on NYNET.

The implementation of OMG CORBA currently used in **xbind** is Orbix. Interworking with other implementations is planned with compatibility among the different ORB implementations achieved through the OMG Internet Inter-ORB protocol (IIOP) [4].

B. **xbind** *Deployment*

As we have already explained in Section I, interoperability is achieved in the binding architecture through the separation of binding algorithms from logical resources. In Table III, we list the hardware and software platform specializations of the VirtualSwitch and VirtualDevice interfaces available in **xbind**.

The first implementation of **xbind** was completed at the beginning of 1995 and its binaries were made available in the public domain in the of summer 1995. Starting with the summer of 1995, we have been deploying **xbind** on the Columbia Campus ATM network by first installing it on three ATM switches. Since then we have been experimenting with **xbind** on the New York State ATM testbed, NYNet, in collaboration with NYNEX S&T and with the Air Force Rome Laboratory. Fig. 10 shows the network topology used in some of the experimentations.

The rapid deployment of **xbind** into NYNET was possible for two main reasons. Firstly, the flexibility of the reservation system for VC's in the VirtualSwitches allowed transparent use of a set of permanent VC's interconnecting NYNET sites. Secondly, in **xbind** the specializations of the VirtualSwitches to the Fore Systems ATM switches were designed to coexist with the existing FORE ATM switch controllers so that the switches participating in the experiments could continue to be fully operational throughout the period of the experimentation. This eliminated the need for a complex reservation system or for limiting our experimentation to off-hours.

Finally, in October 1995 and April 1996 we organized two workshops at Columbia University (see OPENSIG [38]), focusing on technologies related to open signalling such as the Binding Architecture. At present, several research laboratories and universities have been experimenting with **xbind** after installing the binaries on their hosts and switches (see http://www.ctr.columbia.edu/opensig/sites.html).

C. The **xbind** *Video Conferencing Service*

A number of simple multimedia services have been implemented in **xbind**. These include an ATM LAN traffic monitoring service, a connection monitoring service and a video conferencing service that allows setup of multiparty video conference sessions using the connection, device management and transport management services of the broadband kernel services layer. However due to the constraints of space, we will only describe the architecture of the video conferencing service. An overview of its architecture is given in Fig. 11. It consists of four software functional layers.

The lowest layer of software is formed by the BIB, a collection of interfaces that offers an abstract view of resources in the Binding Architecture. These include logical resources like ATM virtual circuit identifiers (VCI's) or physical resources like multimedia devices. Calls are made to these interfaces to bind the underlying resources to create low level services, i.e., broadband kernel services. Upon these, network services are built. Examples of the former include routing and device management, and of the latter, virtual circuits, virtual paths and virtual networks. These services are used to build the multimedia services layer. The multimedia services layer consists of very high level services for multimedia content search, retrieval, distribution and interaction. These include conference management services, multimedia agent-based ser-

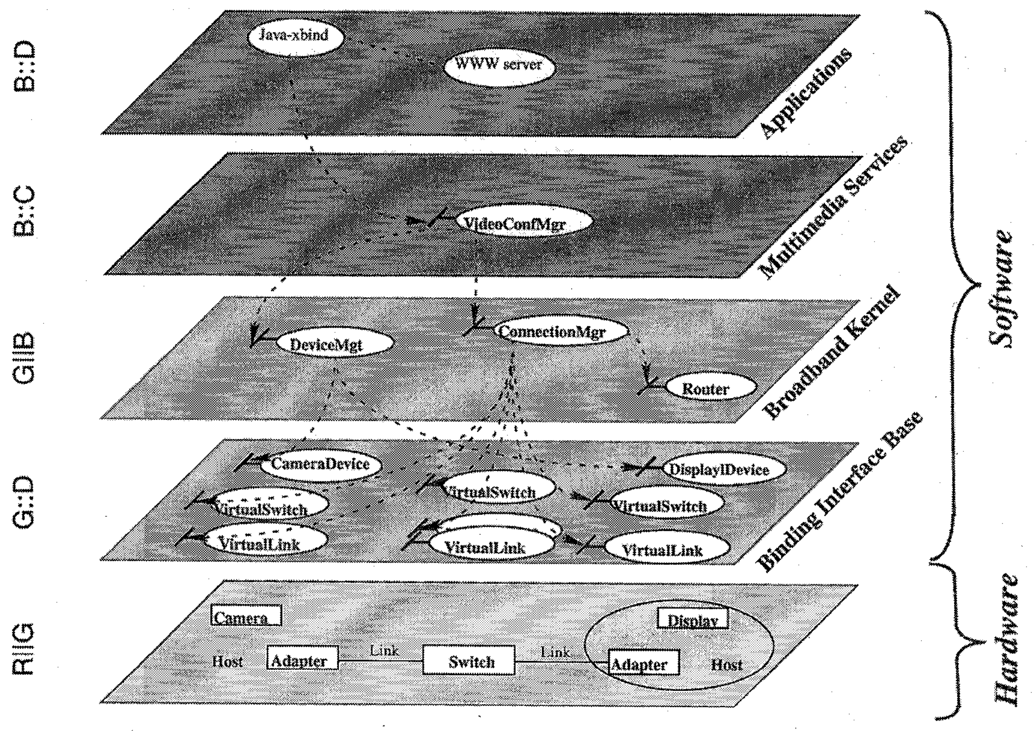


Fig. 11. Java-**xbind** calling sequence.

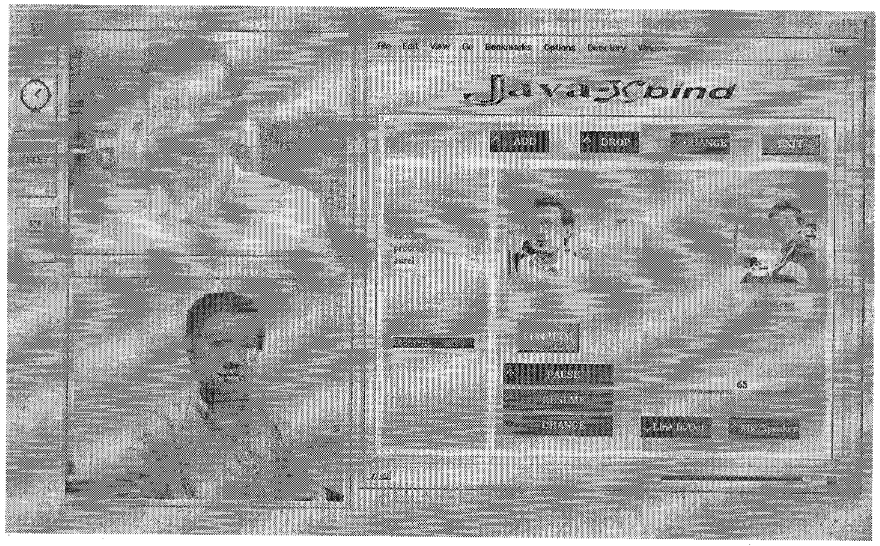


Fig. 12. The Java-**xbind** video conferencing service.

vices, Java [40], [41], and the World Wide Web (WWW) enabled services. At the highest layer of the architecture lie the user level applications which may be WWW browsers or conventional interactive applications.

Located at the application layer, Java-**xbind** [32] is a multiparty desktop teleconferencing service whose functionality is offered through a Java applet loaded with a Java-enabled Web browser. The applet supports typical graphical user interface (GUI) features like scrolled-lists, check boxes, push buttons, bitmaps, etc. that achieve the standard "Motif-like" look and feel. In addition, the applet also contains the logic and data structures for managing a multiparty conference. Fig. 12 shows the applet during the modification of media stream characteristics of a conference session. The interface design assumes that the back-end system provides a directory service that allows easy access to the frequently participating parties. Ideally, the system should be able to perform functions of name mapping and retrieve the actual location of the user and their availability upon a connection request.

To illustrate the relation between the software components of the functional layers of the **xbind** architecture and their interactions, we will describe the calling sequence of the

Java-**xbind** video conference service that we implemented for demonstrating the capability of **xbind**. Note that the discussion here is merely in terms of object invocations. Previously, we described the same service from the perspective of the service architecture.

Referring to Fig. 11, when a client application requests a video connection, the call is translated from the Java applet into a CORBA call. In this case, the call is directed to the video conferencing service entity which in turn makes two calls to the broadband kernel. The first call is to the connection manager to set up an ATM connection between the requested hosts, and the second to the device manager to setup the multimedia devices at the end hosts. The connection manager in turn invokes the routing service to obtain a route for the connection and then invokes a series of calls to the network resources (in this case the VirtualSwitches and VirtualLinks) to reserve the required bandwidth and buffer space. Similarly, the device manager invokes the appropriate series of calls to the multimedia devices to initiate the generation of audio/video streams to the appropriate VCI/VPI's previously reserved. At this stage, the call is complete and the Java-applet is notified of the success or failure of its request.

The communication with the CORBA-based Conference Manager on the back-end is achieved using the orbixweb [14] Java-CORBA integration toolkit made recently available by IONA Technologies. Through this library the applet can perform CORBA calls directly to the ORB resident on the same host of the Web server where the applet was loaded from. This restriction was imposed to the Java applets by the Web browsers because of security issues.

V. Related Work

There has been tremendous interest in the area of service creation and deployment primarily from the direction of the Advanced Intelligent Networks (AIN's) community. In response to this, the Telecommunications Information Network Consortium [3] (TINAC) has recently defined a framework [42] for building TINA-compliant telecommunications services. The primary emphasis of the document is on specifying a broad set of guidelines for building services that must interwork with other TINA defined components. In addition, the framework also identifies and specifies the architecture of specific essential services like subscription, billing, etc.

In terms of standardization efforts of services, the ITU-T has recently specified a series of recommendations [18] for building interworking multimedia conferencing applications. The recommendations specify protocols for transport, conferencing control and multipoint communications as well as application protocols for binary file transfer, still image exchange and annotation. The Multimedia Communications Forum (MMCF) is also working on specifying a set of middleware services [34] for building multimedia desktop collaboration applications that must interoperate across software components implemented by different vendors. Its first series of recommendations have addressed several important aspects of multimedia application interworking including a set of commonly agreed upon application QoS requirements [35].

Historically, one of the earlier groups to investigate the problem from a distributed systems perspective was ANSA [33]. The ANSA architecture specifies a set of components, rules, recipes and guidelines for building scalable distributed applications. The architecture also contains a set of five models for structuring the specification of systems. Although the original target of ANSA was never intended to be the telecommunications industry, a number of its concepts and models were later adopted by the ITU for standardization as part of the open distributed processing reference model [17] (ODP-RM). These recommendations in turn influenced the architecture proposed by TINA [3].

In the area of wireless personal communications systems (PCS), a cluster based distributed call processing architecture has been proposed [20] where call and service processing is delegated away from the switching elements and distributed to specialized servers. This separation of call and service processing from transport and switching simplifies the signalling requirements for mobility management significantly and allows complex PCS services to be more readily implemented.

In the area of object oriented distributed computing technology, work by the Object Management Group (OMG) on CORBA as an industry standard has led to a large number of commercial implementations. Although the initial specifications of CORBA were significantly lacking in the support of services necessary to realize a practical large scale distributed system, its IDL [11] standard enjoys enormous popularity as the language of choice for specifying interfaces. The subsequently proposed OMG object services addresses some of these shortcomings. Recently there have also been a number of proposed extensions to CORBA. Reference [10], for example, proposes real-time extensions to CORBA for building QoS sensitive applications while the SUMO [7] project is investigating the addition of stream interfaces into the CORBA computational model. Others [9] have reported results of benchmarks of commercial implementations of CORBA over ATM networks to test their performance for time sensitive applications.

A number of research projects have also been recently defined for investigating the application of TINA/DPE concepts in telecommunications systems. Among them are the ACTS ReTINA [16] project aimed at developing and demonstrating an industrial-quality DPE, the DCAN [31] project aimed at applying TINA/ANSA philosophies to building a distributed control platform of ATM LAN's and ATM peripherals, and the Magenta [19] project aimed at introducing mobile intelligent agent concepts into TINA.

VI. Conclusion

As seen from the brief survey of the related work, there has been substantial interest and research in the area applying DPE concepts to service creation and deployment in telecommunications. Our work, however, differs from the efforts reported in the literature in two fundamental aspects. Firstly, the emphasis of our approach is on identifying a programming model consisting of a set of states (the BIB) and a set of binding algorithms operating on these states. Secondly, QoS abstractions are an integral component of our model.

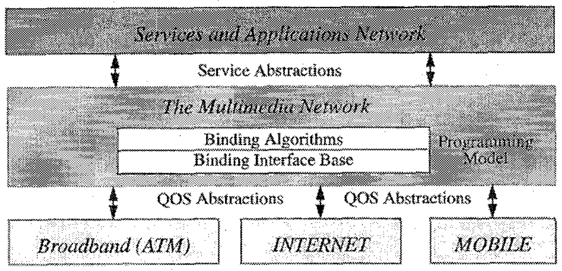


Fig. 13. Realizing interoperability between ATM, Internet, and mobile networks using the binding architecture.

Performance is one of the primary concerns in telecommunications. Therefore, extreme care must be taken at the design stage to ensure that OO techniques like decomposition are not overly applied since they can lead to designs that are too fine grained. Because of these concerns and from our experience with the original BIB design presented in [24], our current BIB reflects only resource state abstractions. Furthermore, explicit abstractions for modeling QoS were introduced so that the issue of performance is factored into our architecture. We believe these two issues to be tantamount in any architecture based on distributed computing for telecommunications.

The primary challenge in the next phase of our work is extending the programming model to the Internet and mobile networks (see Fig. 13) [5]. This requires a better understanding of the scaling properties of the binding architecture. To this end, we believe that the current generation of CORBA development tools are not sufficiently sophisticated to allow scaling without requiring substantial effort on the part of the users. For example, the default location service in Orbix uses a list of hostnames stored on a file as the primary means of locating other ORB servers. Even the set of OMG object services which are being standardized by OMG of which a subset has been recently made available in commercial implementations, are not expected to give appropriate answer to the scaling problem. Other more fundamental problems like partial failures and inconsistency also exist. In this respect, traditional distributed system techniques should in the long run provide answers to these problems so long timeliness and reliability, the cornerstones of telecommunications engineering is not compromised.

ACKNOWLEDGMENT

The authors would like to thank Fore Systems Inc. and NEC USA for their support and help in enabling us to implement the hardware control components of the Virtual Switches on some of their ATM LAN products. The authors would like to also thank the Air Force Rome Laboratory and NYNEX for helping them implement **xbind** on their premises and for supporting the experimentation across NYNET.

REFERENCES

[1] N. G. Aneroussis and A. A. Lazar, "Managing virtual paths on XUNET III: Architecture, experimental platform and performance," in *Proc. 4th Int. Symp. Integr. Network Management*, Santa Barbara, CA, May 1–5, 1995, pp. 370–384.

[2] C. Aurrecoechea, A. Campbell, H. Hadama, and L. Hauw, "A model for multicast in the binding architecture," Center for Telecommunications Research, Columbia University, New York, CTR Tech. Rep. 413-95-19, May 1995.

[3] W. J. Barr, T. Boyd, and Y. Inoue, "The TINA initiative," *IEEE Commun. Mag.*, Mar. 1993.

[4] BNR, Expersoft, IBM, ICL, IONA, SunSoft, ORB 2.0 RFP Submission, OMG TC Document 94.9.32, Sept. 28, 1994.

[5] A. T. Campbell and A. A. Lazar, "**xbind** extensions for QoS controlled mobility," submitted to *2nd Int. Workshop Multimedia Inform. Syst.*, West Point, NY, Sept. 26–28, 1996.

[6] M. C. Chan, H. Hadama, and R. Stadler, "An architecture for broadband virtual networks under customer control," Center for Telecommunications Research, Columbia University, New York, CTR Tech. Rep. 416-95-22, 1995.

[7] G. Coulson, *The SUMO Home Page: Support for Multimedia in Operating Systems*, http://www.comp.lancs.ac.uk/computing/research/sumo/.

[8] J.-P. Deschrevel, "The ANSA model for trading and federation," APM Limited, ANSA Architecture Rep. APM.1005.01, July 1993.

[9] A. Gokhale and D. C. Schmidt, "Measuring the performance of communication middleware on high-speed networks," in *ACM SIGCOMM Conf.*, Stanford University, Stanford, CA, Aug. 1996.

[10] A. Herbert, "CORBA extensions for real-time and interactive multi-media," APM Limited, ANSA Phase III Architecture Rep. APM.1311.02, Oct. 1994.

[11] Hewlett-Packard, IONA, and SunSoft, IDL C++ Language Mapping Specification Joint Submission to the Object Request Broker 2.0 Task Force's C++ Request for Proposals, OMG document number 93-4-4, 1993.

[12] J.-F. Huard, I. Inoue, A. A. Lazar, and H. Yamanaka, "Meeting QoS guarantees by end-to-end QoS monitoring and adaptation," presented at *Proc. 5th Int. Symp. High Performance Distributed Computing*, Syracuse, NY, Aug. 6–9, 1996.

[13] J. M. Hyman, A. A. Lazar, and G. Pacifici, "Real-time scheduling with quality of service constraints," *IEEE J. Select. Areas Commun.*, vol. 9, No. 7, pp. 1052–1063, Sept. 1991.

[14] Iona Technologies Ltd., *Programmers Guide*, Release 1.3, http://www.iona.ie, July 1995.

[15] Iona Technologies Ltd., "Orbix for Java," White Paper, http://www.iona.ie:8000/www/ Orbix/Java/index.html, Feb. 1996.

[16] D. Irlande, *ReTINA—An Industrial-Quality TINA-Compliant Real-Time DPE*, http://media.it.kth.se/SONAH/Acts/AC048.html.

[17] ISO and ITU, *Open Distributed Processing Reference Model, Part 3: Architecture*, ITU-T Rec. X.903, ISO/IEC 107463-3, 1995.

[18] ITU-T, T.120 Data Protocols for Multimedia Conferencing, Proposed Draft, Study Group 8—Contribution_D255/c Revision 2, Mar. 1995.

[19] S. Krause, *The Magenta Project*, http://www.fokus.gmd.de/oks/research/ magenta_e.html.

[20] T. F. LaPorta, M. Veeraraghavan, P. A. Treventi, and R. Ramjee, "Distributed call processing for personal communications services," *IEEE Commun. Mag.*, vol. 33, no. 6, pp. 66–75, June 1995.

[21] M. Laubach, "Classical IP and ARP over ATM," Hewlett-Packard Laboratories, Network Working Group RFC 1577, Jan. 1994.

[22] A. A. Lazar, "Challenges in multimedia networking," in *Proc. Int. Hi-Tech Forum, Osaka'94*, Osaka, Japan, Feb. 24–25, 1994. Available at URL: http://www.ctr.columbia.edu/comet/ xbind/references.html.

[23] A. A. Lazar, "A research agenda for multimedia networking," position paper at the *Workshop on Fundamentals and Perspectives on Multimedia Systems*, International Conference Center for Computer Science, Dagstuhl Castle, Germany, July 4–8, 1994. Available at URL: http://www.ctr.columbia.edu/comet/xbind/references.html.

[24] A. A. Lazar, S. Bhonsle, and K. S. Lim, "A binding architecture for multimedia networks," *IEEE J. Parallel Distributed Comput.*, vol. 30, no. 2, pp. 204–216, Nov. 1995. Also in the *Proc. Multimedia Transport Teleservices*, Vienna, Austria, Nov. 14–15, 1994.

[25] A. A. Lazar, "A binding model for service creation in multimedia networks," in *Workshop High-Speed Networks*, International Conference Center for Computer Science, Dagstuhl Castle, Germany, June 19–23, 1995. Available at URL: http://www.ctr.columbia.edu/comet/xbind/references.html.

[26] A. A. Lazar and G. Pacifici, "Control of resources in broadband networks with quality of service guarantees," *IEEE Commun. Mag.*, vol. 29, no. 10, pp. 66–73, Oct. 1991.

[27] A. A. Lazar, K. S. Lim, and F. Marconcini, "Binding model: Motivation and description," Center for Telecommunications Research, Columbia University, New York, CTR Tech. Rep. 411-95-17, June 1995. Available under URL: http://www.ctr.columbia.edu/comet/ xbind/xbind.html.

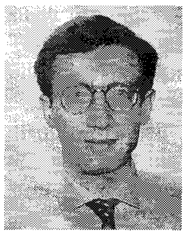
[28] ——, "The binding interface base," Center for Telecommunications Research, Columbia University, New York, CTR Tech. Rep. 412-95-18, June 1995. Available under URL: http://www.ctr.columbia.edu/comet/xbind/xbind.html.
[29] A. A. Lazar and F. Marconcini, "Toward an open API for ATM switch control," Center for Telecommunications Research, Columbia University, New York, CTR Tech. Rep. 441-96-07, Feb. 1996. Available under URL: http://www.ctr.columbia.edu/comet/xbind/xbind.html.
[30] A. A. Lazar and K. S. Lim, "Programmability and service creation for multimedia networks," presented at *Proc. 5th Int. Symp. High Performance Distributed Computing*, Syracuse, NY, Aug. 6–9, 1996.
[31] I. Leslie, *Distributed Control of ATM Networks*, http://www.ansa.co.uk/DCAN.
[32] K. S. Lim and F. Marconcini, "Java-xbind—A Java-enabled xbind video conferencing application," Columbia University, New York, CS 6998 Project Report, http://www.ctr.columbia.edu/~franco/java-xbind.html, Jan. 1996.
[33] R. V. D. Linden, "An overview of ANSA," APM Limited, ANSA Architecture Rep. APM.1000.01, July 1993.
[34] MMCF, MMCF Middleware, MMCF/95-005 DRAFT 3.2, Suite 201-931, Brunette Avenue, Coquitlam, BC, Canada, V3K 6T5.
[35] MMCF, *Quality of Service*, MMCF/95-010 Approved Rev 1.0, http://www.mmcf.org/ QoS9510.zip.
[36] P. Newman, R. Hinden, E. Hoffman, F. C. Liaw, T. Lyon, and G. Minshall, *General Switch Management Protocol Specification*, Version 1.0, Ipsilon Networks, Inc., Feb. 1996.
[37] Object Management Group (OMG) and X/Open, *The Common Object Request Broker: Architecture and Specification*, Revision 1.2, Dec. 1993.
[38] OPENSIG: http://www.ctr.columbia.edu/opensig/opensig.html.
[39] Project **xbind**: http://www.ctr.columbia.edu/comet/xbind/xbind.html.
[40] Sun Microsystems Inc., "The Java Language Environment," White Paper, Mountain View, CA, Oct. 1995.
[41] Sun Microsystems Inc., *The Java Language Specification*, Version 1.0 Beta, Mountain View, CA, Oct. 1995.
[42] TINA-C, *Service Architecture Version 2.0*, Document No. TB_MDC.012_2.0_94, TINA-C, Mar. 1995.

Aurel A. Lazar (S'77–M'80–SM'90–F'93), for a photograph and biography, see this issue, p. 1212.

Koon-Seng Lim received the B.Sc. degree in computer science from the National University of Singapore (NUS), Singapore, in 1991, and the M.S. degree in electrical engineering from Columbia University, NY, in 1996.

He joined the Institute of Systems Science as a full time software engineer in May 1991 and he has been working in the area of performance management since then. In 1994, he was awarded a scholarship from the National University of Singapore to pursue a Ph.D in electrical engineering at Columbia University, NY. He is currently with the COMET Group at the Center for Telecommunication Research at Columbia University working on architectural issues of multimedia networks.

Mr. Lim was the recipient of the Halbrecht Associates Book Prize awarded in 1991 by the National University of Singapore for his work on the performance analysis of multimedia backbone FDDI LAN's in his final year dissertation (http://www.ctr.columbia.edu/ ~koonseng).



Franco Marconcini received the B.S. degree in computer science from the University of Milan, Italy, in 1990, and the M.Sc. degree in electrical engineering from Columbia University, NY, in 1996.

After graduation, he worked as software engineer designing and developing distributed information systems for commercial applications. His work experience includes the development of quality systems compliant with the ISO 9002 standard. He has research interests in support systems for multimedia services creation and in the market-based analysis of the driving forces that will enable the global deployment of multimedia services. In 1993, he was awarded a Postgraduate Fellowship from the University of Milan and joined the COMET Group at the Center for Telecommunication Research at Columbia University as a Visiting Scholar. He is currently working as a software designer at Columbia University. In the COMET Group, he played a major role in the design, development, and deployment of the ATM network services of the "xbind" project as well as in the experimentations over the NYNet testbed. He represents Columbia University in the NYNet Technical Committee (http://www.ctr.columbia.edu/ ~franco).