A Binding Architecture for Multimedia Networks

Aurel A. Lazar[†], Shailendra K. Bhonsle[*] and Koon Seng Lim[*]

Abstract:

An open architecture that achieves seamless binding between networking and multimedia devices is proposed. The building blocks of the binding architecture consist of a set of interfaces, methods and primitives. The former abstract the functionalities of multimedia networking devices and are organized into a binding interface base. The methods and primitives are invoked for implementing binding applications. The binding architecture is embedded into a reference model for multimedia networking architectures that supports a clean separation between binding interfaces and binding algorithms. Communication between the interfaces of the architecture is supported by CORBA. Public interfaces in the binding interface base are specified using CORBA IDL. The architecture is illustrated with a simple connection management algorithm and an example of computational binding.

1. Introduction

We start by presenting the motivation for our work. This is followed by a review of some of the pertinent literature and a description of the methodology employed for designing a binding architecture for multimedia networks.

1.1 Motivation

Binding is the process of associating (interconnecting) different components of a system. The binding architecture of multimedia networks dictates how its entities are modeled and how these entities are associated with each other in order to provide the user of a service with a “holistic” picture. The architecture itself consists of a binding interface base and binding algorithms. Binding architectures and applications for networking and multimedia computing have been developed for the most part independently. As a result, there is no uniform terminology in these fields: connection management, binding, signalling protocols, etc. are words often used interchangeably.

Connection management in telephone networks is resolved by defining a User/Network Interface (UNI) and a Network/Node Interface (NNI). These interfaces are realized through the Q.93b and CCSS #7 (Common Channel Signalling System), respectively. International standards bodies are considering the CCSS #7 together with the Q.93b interface as the basis for signalling in broadband networks. There are a number of problems with this solution, however.

†. Department of Electrical Engineering and Center for Telecommunications Research, Columbia University, New York, NY 10027-6699, email: aurel@ctr.columbia.edu

*. Institute of Systems Science, National University of Singapore, Singapore, 0511.

The UNI and the NNI concepts, introduced in the 60s, rightly recognized that the Customer Premises Equipment (CPE) had a low level of intelligence in comparison with the switching equipment. That has now changed as the customer might possess the latest powerful workstation or parallel machine. In fact, the customer equipment is often at least as intelligent as the switching controllers. Broadband networking requirements for defining and manipulating virtual networks and multicasting are readily modeled as high level objects. It is natural, therefore, to provide higher level language constructs in describing connection management operations. Note that in this context, the UNI/NNI model is akin to a *low level* programming language. Development of signalling protocols based on object-oriented call models does not change this basic assessment.

The Internet community has developed connection management capabilities as part of the TCP/IP suite. Currently, in order to support extensions of the architecture to multimedia services, reservation protocols are being investigated. An evolutionary path towards interworking with future broadband networks is not yet available.

The Interactive Multimedia Association (IMA) is considering proposals for interoperability of distributed multimedia systems. While the networking aspects have not yet been considered, CORBA and IDL have gained wide acceptance for implementing any such systems. The IMA recommended practice is likely to gain wide acceptance in the computer industry.

While there has been considerable work in the individual areas of signalling protocols, object based network architectures and on addressing the issue of interworking of multimedia devices, there has been little work on defining a *seamless architecture* that will integrate all these concerns. It is our belief that such an architecture is needed to support a multitude of applications such as connection management, distributed computing, signalling protocols, etc. The need to design and implement a binding architecture as will be discussed in this paper has been first recognized in [19].

1.2 Related Work

In [11] the facilities required to control and manage multiservice applications in ATM networks are examined in detail. The requirements necessary for dealing with diverse service configurations are defined. As such, issues pertaining to multimedia services and in particular, Quality of Service (QOS), are not addressed.

Another perspective which focuses on specifying the service description of a system rather than its individual components was proposed in [14]. Here, the work concentrates on developing service primitives and their sequence of invocations at service access points for multimedia multiuser services. Architectural and implementation issues were deliberately omitted.

Defining a signalling protocol capable of supporting complex multimedia services is central to the investigations in [26]. Although object-oriented in nature, the model presented limited itself to representing only call related aspects of the network and omits any discussion of the architectural aspects of how these may be realized and integrated within an overall architecture.

The ATM Forum is investigating the applicability of a subset of the Q.93b UNI [7] for broadband applications in the local environment. [12] is representative of this work. Work describing the NNI and the CCSS #7 can be found in [8].

In the areas of protocols that guarantee QOS, one of the more promising developments has been the proposal of two reservation-based protocols for the Internet. The RSVP protocol [29], is a receiver-oriented simplex protocol that can accommodate heterogenous receivers in a multicast group and allow dynamic changes in group membership through maintenance of a ‘soft-state’ in each node. As opposed to this, the sender-initiated approach is adopted by the ST-II protocol [28]. The protocol uses multiple simplex reservations to create stream-based multicast trees. Since reservation is on a per-tree basis, ST-II cannot accommodate heterogeneous receivers.

The initial version of the Touring Machine project [5] focused on providing a simple, point-to-point desktop video communication service. The second generation system provides APIs for application developers to aid widespread deployment. In both instances, the emphasis was on building a workable prototype so that showcase multimedia applications can be developed rather than on building a generic architecture.

On a different track and much wider in scope, the work by the TINA consortium [3], [4] centers on the development of an Information Networking Architecture that would bring together distributed computing, telecommunications and management standards into a single framework. TINA has not yet addressed the problem of interworking with multimedia devices. The same applies to the substantial contributions put forth by ANSA [2] and the follow up ODP architecture [25].

The methodology developed for binding multimedia objects in Multimedia Systems Services (MSS) [15] is based on modern foundations of distributed algorithms and software engineering [1], [9]. The MSS proposal is object-oriented. The interfaces are specified in the IDL interface definition language [17]. In order to support interaction among distributed interfaces, MSS depends upon the Object Management Group’s (OMG) Common Object Request Broker Architecture (CORBA) [10].

1.3 Methodology

An ad-hoc approach for interconnecting a multimedia system such as MSS to a broadband network specified via an UNI would be to present it with the Q.93b interface. This approach, however, exhibits the limitations already mentioned in section 1.1. We, therefore, advocate a solution based on a different modeling paradigm.

In this paradigm, the binding architecture and binding applications are clearly separated. The architecture itself is *open* and hence possibly subject to standardization efforts. The binding applications, however, might not be. Binding entities in the architecture are modeled as communicating objects. As in MSS, CORBA provides the high level location independent communication facilities. This allows for a seamless binding environment between the network and the multimedia resources. Overall, binding operations exhibit a much lower level of complexity.

This paper is organized as follows. In section 2 the modeling framework provided by the Extended Integrated Reference Model is briefly reviewed and binding within this model discussed. The architectural model of the binding architecture is presented in section 3. The relationship between the binding architecture and other ongoing work is also discussed. The binding interface base, including the interface inheritance diagram and some of the interface definitions are presented in section 4. Section 5 describes the binding methods and primitives. Examples of binding applications are given in section 6. Conclusions and future directions are given in section 7.

2. Modelling Framework

In this section the framework for binding architectures provided by the Extended Integrated Reference Model (XRM) is presented. The XRM is discussed in section 2.1. In section 2.2 the positioning of binding within the XRM is described.

2.1 The Extended Integrated Reference Model (XRM)

In parlance of network architectures, Figure 1 is an abstract representation of the *Extended Integrated Reference Model (XRM)* [19]. The XRM models the communications architecture of broadband networks and multimedia computing platforms. The foundations for the operability of multimedia computing and networking devices is the same. Both classes of devices can be modeled as producers, consumers and processors of media. The only difference appears to be in the overall goal that a group of devices is set to achieve in the network or the multimedia platform.

The restriction of the XRM to broadband networks is called the Integrated Reference Model (IRM) [18]. The IRM incorporates monitoring and real time control, management, communication, and abstraction primitives that are organized into the *Traffic Control Architecture*, the *Management Architecture*, the *Information Transport Architecture* and the *Telebase Architecture*, respectively. The subdivision of the IRM into the Management and the Traffic Control Architectures on the one hand, and the Information Transport Architecture on the other, is based on the principle of separation between controls and communications. The separation between the Management and the Traffic Control Architecture is primarily due to the different time-scales on which these architectures operate.

The Integrated Reference Model is organized into five planes that model the above architectures (Figure 1). The Management Architecture resides in the network management or N- plane, and covers the functional areas of network management, namely, configuration, performance, fault, accounting and security management. Manager and agents, its basic functional components, interact with each other according to the client-server paradigm. The Traffic Control Architecture consists of the resource control, or M-, and the connection management and control, or C-, planes. The M-plane comprises the entities and mechanisms responsible for resource control, such as cell scheduling, call admission, and call routing; the C-plane those for connection management and control. The Information Transport Architecture is located in the user transport or U-plane, and models the protocols and entities for the transport of user information. Finally, the Telebase Architecture resides within the Data Abstraction and Management or D-plane, and implements the principles of data sharing for network monitoring, control and communication primitives, the functional building blocks of the N-, M-, and C- and U-plane mechanisms. (A mechanism is a functional atomic unit that performs a specific task, such as setting up a virtual circuit in the network [24].)

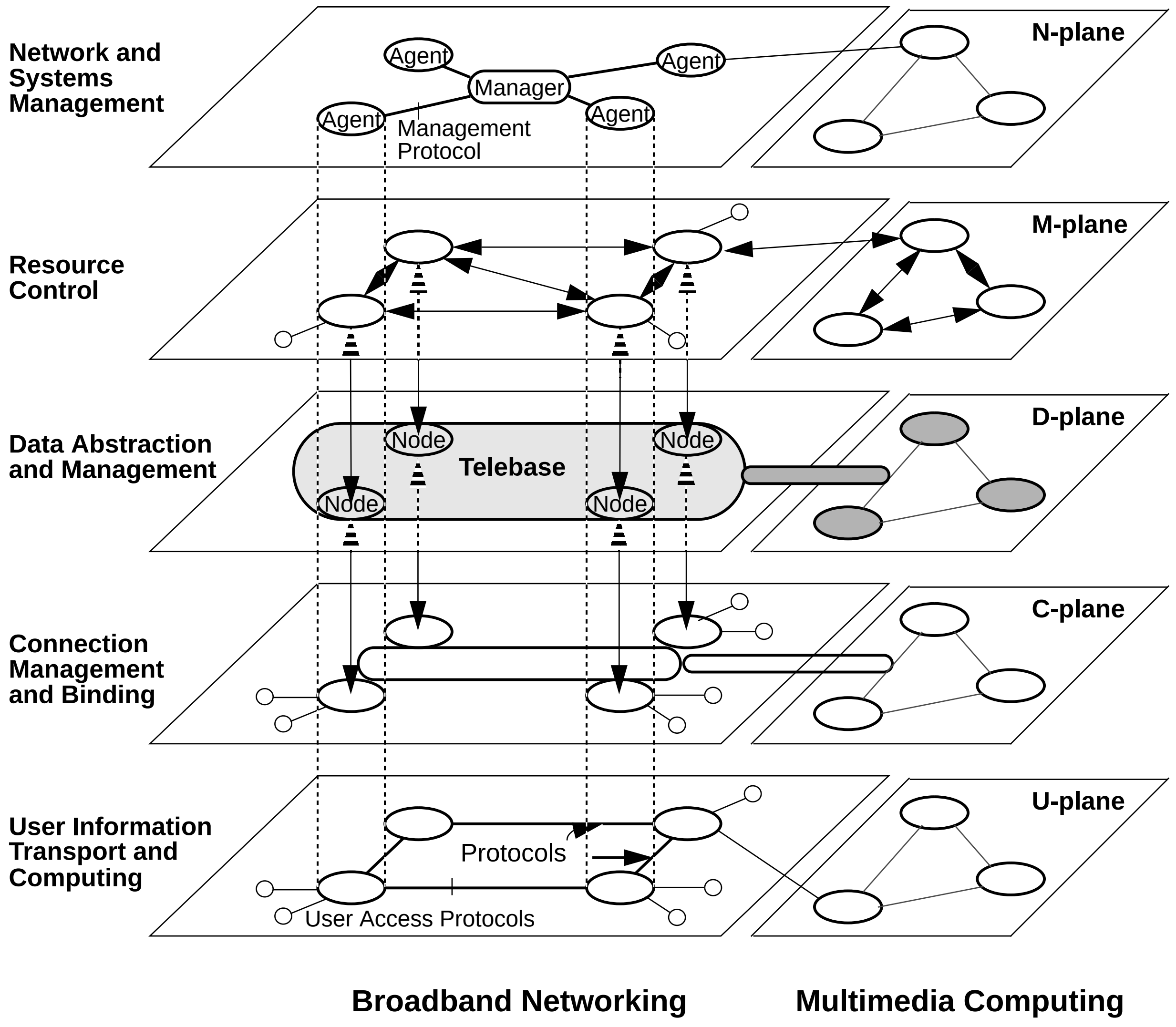

Figure 1. The Extended Integrated Reference Model (XRM).

The restriction of the XRM to the multimedia computing platform has a similar functionality as the IRM. The N-plane includes system management functionality, and the M-plane includes process scheduling, memory management, routing (when applicable), admission control and flow control. The D-plane also contains objects modeling multimedia devices, the C-plane binding functionality, and the U-plane transport of user information within the Customer Premises Equipment.

2.2 Binding within the XRM

Binding requirements arise in each of the planes of the XRM. However, dynamic binding requirements are particularly demanding in the C- and M- planes. In order to better understand and fullfil these requirements, a *separation principle* between the binding architecture and applications running on top of it is defined. This separation principle gives a clear focus towards what should be and what needs to be standardized within the XRM. It also allows us to take a very general point of view towards binding.

The binding architecture resides in the M-,D- and C-planes of the XRM. Specifically, the binding interface base resides in D- plane and the binding algorithms execute from within the M- and C- planes. The binding architecture represents a software environment on top of which all the binding applications execute. Scalability of this architecture is achieved with a distributed object-oriented design. Binding interfaces can be added as the need arises.

Binding applications run on top of the binding architecture. Examples of binding applications arise in connection set up for broadband networks, distributed systems implementing synchronization protocols, resource allocation protocols such as those intended for the Internet, multimedia computing platforms, etc. New binding applications can be added without changing the underlying binding architecture. Note that, several proprietary binding algorithms supporting various applications can operate at the same time.

3. The Binding Architecture

An overview of the binding architecture on the system level is given below. Section 3.1 presents the architectural model. In section 3.2 the relationship between our binding architecture and the MSS architecture is described. Finally, a brief comparison with the OSI Network Management and ODP architecture is given.

3.1 Architectural Model

The binding architecture proposed here is open: all multimedia networking entities participating in the binding process are modeled as communicating objects with well defined interfaces that can be externally invoked. Interface methods and some global primitives are used for these invocations. Binding algorithms operate upon these interfaces.

The interfaces are realized as objects modeling resources such as switches, links, multimedia devices, etc. All interfaces reside in a data repository called the Binding Interface Base (BIB). More abstractly, the BIB provides multimedia networking abstractions for producers, consumers, and processors of media. Interfaces in the BIB are defined using the CORBA IDL (Interface Definition Language) specification language. The BIB containing all binding interface instances (called binding objects) reside in the D-plane of the IRM. CORBA provides naming facilities to locate interface implementations and invoke methods. A factory is used to instantiate an interface and one of the embedded methods within the interface can be invoked to delete the interface instance.

Public methods are visible to different “multimedia networking clients” who can invoke them. Multimedia networking clients are “clients” in CORBA sense. For example a binding algorithm that invokes binding interfaces is a multimedia networking client as is the user of a “service” that invokes the BIB interfaces and binding algorithms. Scalability of the binding architecture is readily achieved by adding new interfaces or by upgrading existing ones. The addition of new binding algorithms can also be easily accomplished.

The components of the binding architecture consisting of the binding interface base and the binding algorithms are depicted in Figure 2. This figure shows the distributed nature of BIB interactions, and the distributed interactions amongst different binding algorithms.

3.2 Relationship to Binding for Multimedia

The reader has probably recognized by now a number of similarities between our binding architecture proposal with the Multimedia Systems Services [15] platform considered by the Interactive Multimedia Association (IMA). Recall that MSS constitutes a framework of “middleware” — system software components lying in the region between the generic operating system and specific applications. Its goal is to provide an infrastructure for building multimedia computing platforms that support interactive multimedia applications dealing with synchronized, time-based media in a heterogeneous distributed environment. It is under evaluation by the IMA and is expected to become a recommended practice within the computer industry.

How does the MSS framework fit into the XRM? Here we distinguish between facilities for creating and removing objects as well as binding operations. In MSS a number of interfaces have been defined to enable both the creation and destruction of objects that participate in the binding process. Creation and destruction operations are D-plane native. Binding algorithms on the other hand are C- and M- plane visible. Sizing of virtual resources derived from QOS requirements is exported through M- and N-plane interfaces.

What are the differences between our architecture and the architecture of the MSS? We believe that only the binding interface base should be standardized although there might be a need to standardize some of the binding algorithms and applications for higher level interoperability. We feel that management and control tasks, such as QOS control and management that the current MSS architecture proposes to fullfil, are best modeled as M- and C- plane binding algorithms. The fundamental abstractions that these algorithms operate upon are modeled as BIB interfaces.

3.3 Relationship to the OSI Network Management Architecture and ODP

There are also important conceptual similarities between our binding architecture and the OSI management architecture [6]. As in OSI management, we propose to have an object-oriented model for the entities of interest, a standard communication support and a well defined set of interfaces that support basic (binding) operations. Both, the OSI management architecture and the binding architecture, are separated from the management and binding applications, respectively. There are of course a number of differences, the main one being that the OSI management architecture is entirely centralized whereas the binding architecture discussed here is entirely decentralized. Another major difference is the time scale on which these architectures operate.

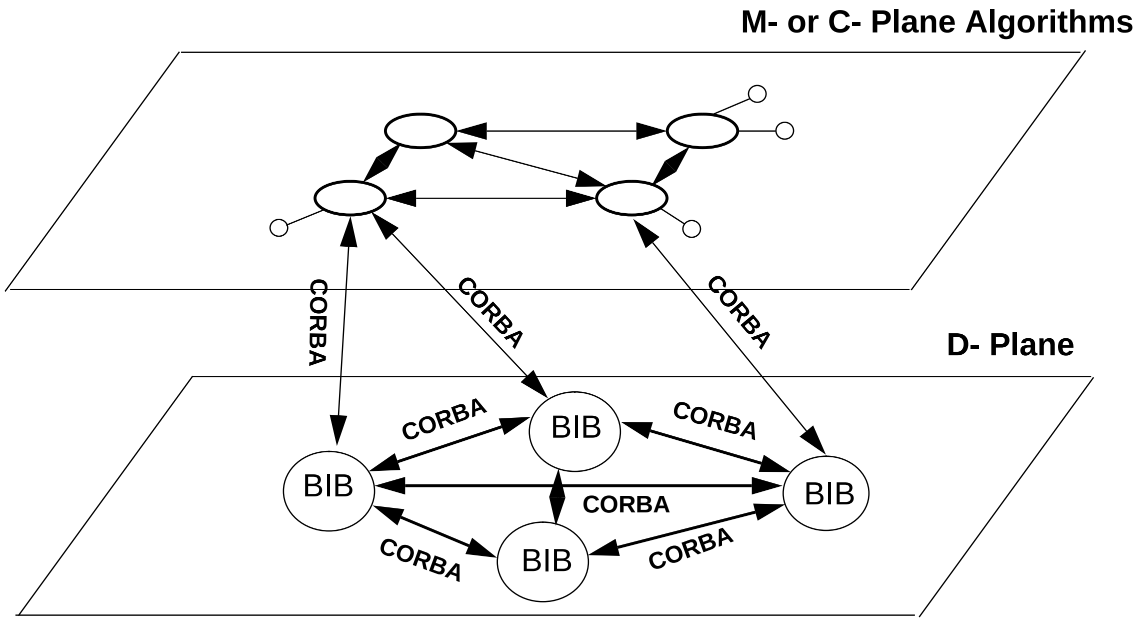

Figure 2. Distributed Binding Algorithm Interacting with Distributed BIBs

We note a number of similarities with the ODP architecture as well. For example, the ODP information model, for expressing the meaning of information and information processing tasks is analogous to our definition of the D-plane; the computational model for expressing functional decomposition of the systems into distributable units with well-defined interfaces corresponds directly to our standard object-based implementation; the engineering model for describing components and structures needed in support of distribution is expressed in our proposed use of RPC and CORBA both of which are stable and well accepted; and finally the technology model for describing the makeup of a system in terms of components that conform to appropriate standards fits in nicely with our emphasis on CORBA and interworking with MSS.

4. The Binding Interface Base

In this section the inheritance diagram underlying the BIB is discussed. We briefly present the structure and the semantics of some of the BIB interfaces.

4.1 Interface Inheritance Diagram

The interface inheritance diagram for the BIB is depicted in Figure 3. In the future specialized interfaces will inherit from the generic interfaces, thus extending the diagram horizontally. When new interfaces, such as those shown with shaded lines are added to the architecture, the diagram is simply extended vertically. As with the OSI network management MIB, the BIB resides in the D-plane of IRM. Both are integral parts of the architecture of the Telebase.

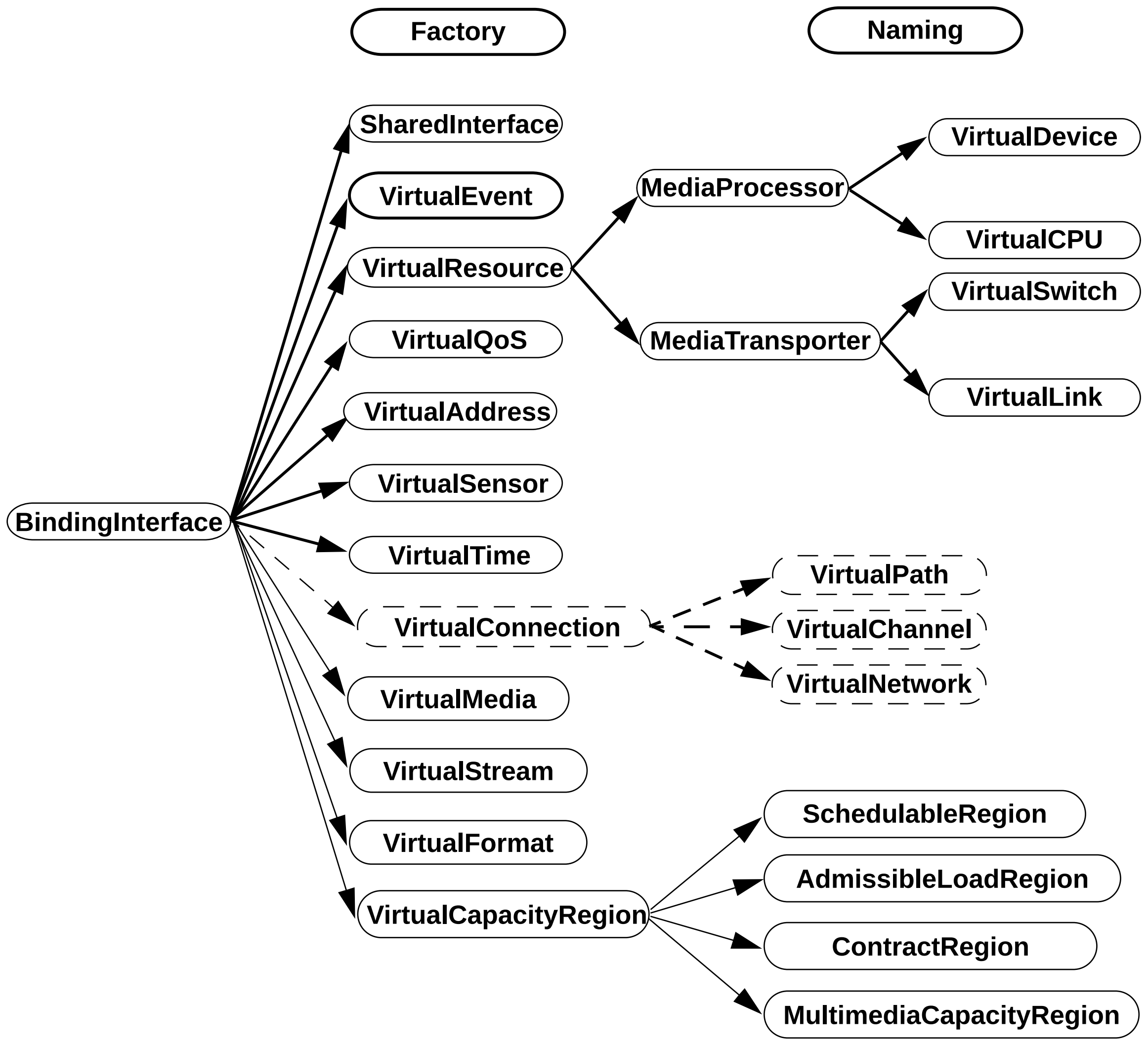

Figure 3. Interface Inheritance for Binding Interface Base

The BIB interfaces are defined in CORBA’s [10] Interface Definition Language [17]. Adopting CORBA terminology, they are realized as object-implementations (servers). An instantiation of an interface is an object within these object-implementations. Since CORBA’s view of interfaces is analogous to a “class” in object-oriented programming and since we only employ an object-oriented implementation of interface definitions, the term "class" will be used interchangeably with "interface" and "object" interchangeably with the notion of an "instantiation of an interface".

Interface implementations contain many object management specific public interfaces like creation and deletion of objects. These will be of no concern here. Our *primary goal* is to show what are the various public interfaces pertinent to binding operations and what are the *local states* that they manipulate. The following section briefly describes the essence of our design. A more detailed specification is available in [21].

4.2 Interface Definitions

There are two standard interfaces namely ***Factory*** and ***Naming.*** They are both derived from standard CORBA specifications. The ***Factory*** is a generic component that instantiates and deletes other objects remotely. The ***Naming*** module deals with "CORBA specific" naming conventions as well as registration and retrieval of interfaces as it is necessary to locate and invoke the right object in the distributed environment.

Interface inheritance is a static (compile-time) relationship between the various interfaces that the BIB contains. We describe here the rationale for some of the interface definitions.

A key interface that the binding interface base provides is the ***VirtualResource.*** Inherited interfaces from ***VirtualResource*** model all the physical resources that are present in the multimedia networking architecture. Note that at runtime only the physical resources that are present within the "scope" of a node's BIB will have corresponding ***VirtualResource*** objects. The ***VirtualResource*** is subclassed into ***MediaProcessor*** and ***MediaTransporter.*** The relative positioning of ***MediaTransporters*** and ***MediaProcessors*** is shown in Figure 4.

The ***MediaProcessor*** models all producers, consumers and processors of media. The generic classes ***VirtualCPU*** and ***VirtualDevice*** are derived from ***MediaProcessor***. The ***VirtualDevice*** is a consumer or producer of multimedia information. It represents exactly the same devices as considered by MSS [15]. Note, however, that MSS does not consider the distinction between a ***MediaProcessor*** and ***MediaTransporter*** as essential for their architecture.

The ***MediaTransporter*** is subclassed into ***VirtualSwitch*** and ***VirtualLink***. A physical switch consists of a set of multiplexers interconnected through a switch fabric with an associated control unit. The ***VirtualSwitch*** represents the "local" control elements of a switch fabric. In our model these control elements are distinct from the control elements and the intelligence associated with the output multiplexers. The multiplexers located at the output ports of the switch are modeled by the ***VirtualLink*** interface. Thus, the ***VirtualLink*** models the cell and call resources consisting of output buffers and links whereas the ***VirtualSwitch*** models the resources associated with the VP/VC routing tables. The distinction between the ***VirtualSwitch*** and the ***VirtualLink*** is depicted in Figure 5.

Another important interface contained in the BIB is the ***VirtualCapacityRegion***. This interface characterizes the capacity of multimedia networking resources with QOS guarantees. With every ***VirtualResource*** is associated a ***VirtualCapacityRegion.*** Its state is typically represented by an ***OperatingPoint*** within this region. The axes of the ***VirtualCapacityRegion*** represent the QOS classes (which in turn may be related to QOS parameters). Different resources may have their ***VirtualCapacityRegion*** axes labeled differently. For example with a ***VirtualCPU*** a ***MultimediaCapacityRegion*** is associated whereas a ***VirtualLink*** is characterized by its ***SchedulableRegion.*** It is the responsibility of the underlying resource control mechanisms to map "service" class information into appropriate traffic class information and vice versa. The ***VirtualQoS*** interface is provided to do this translation.

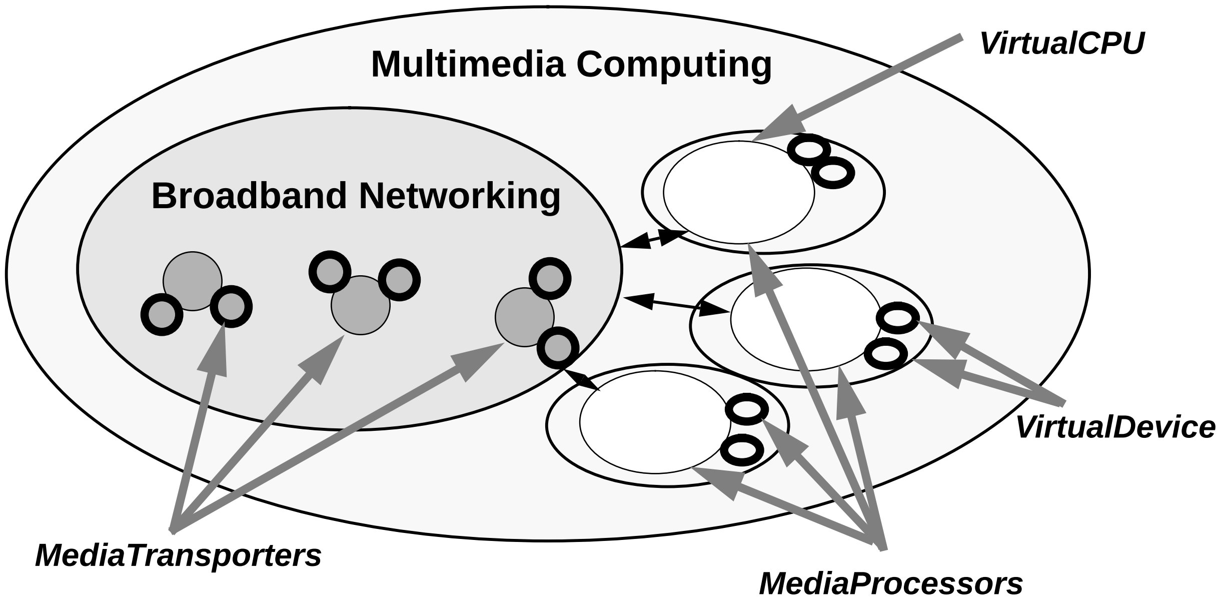

Figure 4. Relationship between *MediaProcessor and MediaTransporter*

The ***VirtualAddress*** interface provides translation of various types of addresses (ATM-Forum addresses, Generic addresses, CORBA related addresses) into each other, whenever possible. Different inherited interfaces may add/extend address translation facilities of this interface.

There are two generic interfaces that can be inherited from to provide event forwarding services as well as "monitoring/sensing" parameters of interest. ***VirtualEvent*** provides for the definition, registration and forwarding of events to any number of client algorithms. The ***VirtualSensor*** interface provides for monitoring activities of interest within the binding architecture. Both of these interfaces are for "resource" control purposes and hence are "different" in purpose than N-plane specific management event forwarding and monitoring capabilities. Since they are going to be used by fast time-scale control algorithms, they must be kept lightweight and efficient and their use must be guarded by some overall real-time constraints.

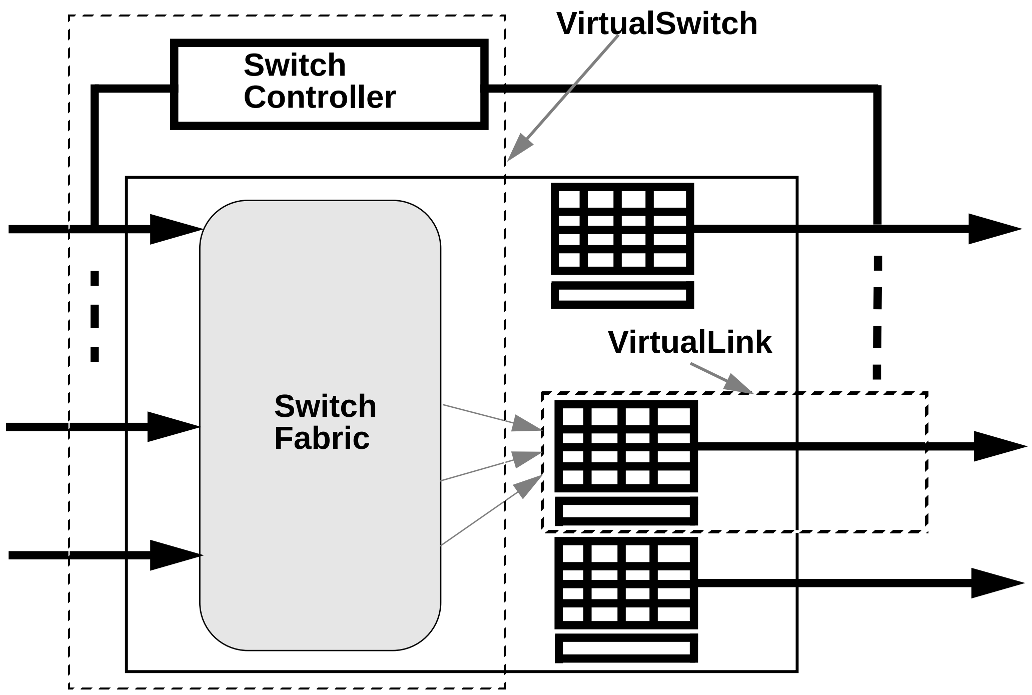

Figure 5. VirtualSwitch and VirtualLink

Real-time requirements that the various binding algorithms may have are supported by the ***VirtualTime*** interface that provides a local clock. This interface supports both synchronous and asynchronous timeouts. Time related ***VirtualEvents*** may be associated with the ***VirtualTime*** interface.

There are three media related interfaces: ***VirtualMedia, VirtualStream, and VirtualFormat***. ***VirtualMedia*** defines the properties of the various multimedia: video traffic, CD Quality traffic, etc. Associated media format and stream-properties have been abstracted into two related interfaces, ***VirtualStream*** and ***VirtualFormat,*** respectively. These interfaces are defined for use by ***VirtualResource***, especially ***MediaProcessor,*** and also for annotating ***VirtualCapacityRegion*** axes.

SharedInterface is an interface that allows for generically manipulating other interfaces. It is also capable of manipulating “state expressions” to answer certain resource state specific queries. We are currently in the process of defining this interface.

CORBA integrates the normal remote procedure call (RPC) paradigm with “event” handling facilities. It provides standard RPC protocol related events and one can also define events through using its “exception” and “raises” primitives. In [21] it is shown how interface definitions incorporate standard as well as user defined events.

5. Binding Methods and Primitives

In this section binding methods and primitives are discussed. There is a distinction between methods and primitives. The former are interface methods and are invoked by CORBA clients. Apart from methods which are associated with individual interfaces, there are also global “primitives” of the binding architecture. These primitives are implemented as interface methods as depicted in Figure 6. Primitives are global atomic units of execution.

There is a need to identify and classify separately important methods that *must be present* in the BIB interfaces. Specifically since we are controlling certain devices by changing their states, there is a need for imposing consistency of usage by “different” algorithms that coexist in the multimedia networking architecture. In this section we briefly describe the essential elements of these. Details can be found in [21].

```
1. getState(in BI bObject, out State biState);
2. setState(in BI bObject, in State newState);
3. getReservationState(in BI bObject, out ReservedState rState);
4. setReservationState(in BI bObject, in State newState);
5. commitReservedState(in BI bObject);
6. evalStateExp(in TypeExp type, in StateExp stateExp, out Boolean result, out
   State rState);
7. parseStateExp(in TypeExp type, in StateExp stateExp, out long result);
8. executeMethod(in BI bObject, in FunctionPtr func, out sequence of OutPars
   outparams, out RetType returnedValues);
9. queryBI(in BI bObject, out sequence of BAttributes bAttrs);
```

Figure 6. Primitives Used for Binding

The public methods are used for porting the BIB onto physical devices, bootstrapping, and attaching appropriate transport protocols for supporting CORBA RPC communications. They consist of atomic primitives for getting and setting states.

5.1 Binding Primitives

There is a set of important primitives that must be present either as methods in ***VirtualResource*** and many other derived interfaces or as methods in ***SharedInterface***. These primitives essentially locate and manipulate states, reserved states, and other attributes of an interface.

Primitives for State

The concept of current state has been associated with all interfaces that model physical resources. The primitives ***getState()*** and ***setState()*** extract the current state and set the current state, respectively. For example separate state interfaces can be attached in the same BIB for a ***VirtualLink***. It is the responsibility of ***getState()*** and ***setState()*** primitives to maintain the consistency between various definitions.

Primitives for Reserved State

The reserved state is an important concept in our architecture. It indicates certain resources that have been locked by a “client”. Such locking might occur for enforcing consistency requirements on the part of the client. There are two methods for setting and getting reserved states: ***getReservationState()*** and ***setReservationState()***.

EvalStateExp() Primitive

This primitive takes as an input a ***StateExp*** and evaluates it to either a ***Boolean*** (true or false) or to another ***State***. A ***StateExp*** could be either a predicate or an expression involving state variables and state operators. It is important to note that the ***StateExp*** may contain an evaluation of expressions with variables outside the scope of a given BIB. Note also that there is a limit to the expressive power of ***StateExp*** as it is not intended that binding algorithms evaluate very complex functions.

ExecuteMethod() and QueryBI() Primitives:

These primitives provide means to implement generic “method execution” and getting the values of certain attributes indirectly.

6. Examples of Binding Algorithms and Applications

The example given in section 6.1 describes how the binding architecture can support connection management whereas the example in section 6.2 describes how a distributed computing application can be supported.

6.1 Connection Management with QOS Guarantees

In this section we briefly describe an example of a simple connection management algorithm that can be built on top of the binding interface base of a broadband ATM network. The emphasis of the exercise is not on the connection management algorithm itself, but on illustrating how network binding can be implemented within our architecture.

Figure 7 shows the interface inheritance diagram for this example. We define an interface called ***VirtualConnection***. A physical connection exists as a set of interconnected entries in appropriate tables (e.g., routing tables). A corresponding interface in the BIB represents these physical entities. The ***VirtualConnection*** interface is subclassed into ***VirtualPath, VirtualChannel,*** *and* ***VirtualNetwork*** modeling the “virtual path”, the “virtual channel” and the “virtual network”, respectively. These interfaces also provide methods to setup/release connections on ATM networks. The realizations of ***VirtualPath, VirtualChannel,*** *and* ***VirtualNetwork*** as objects are called network applications. Applications with QOS guarantees are called services.

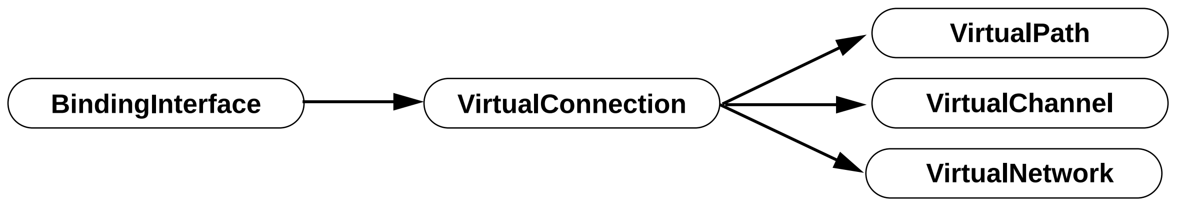

Figure 7. Interface Inheritance Diagram for Connection Management.

Connection management algorithms create the above network services (i.e., the ***VirtualPath, VirtualChannel,*** *and* ***VirtualNetwork*** objects). Based in the C-plane, these algorithms operate on objects residing in the D-plane and interact with M-plane resource control algorithms only through D-plane objects. A "connection manager" in the C-plane can implement any connection management algorithm (in addition to the ones that adhere, e.g., to the Q.93b or SS7 specifications).

Connections with guaranteed QOS (i.e., network services) can be established because the states of local D-plane interfaces are bound to multimedia networking devices with QOS guarantees as discussed in a separate paper [22]. QOS parameters are used to define service classes [7]. As already mentioned, the mapping to traffic classes [23] or other QOS parameter specifications is the prime responsibility of the ***VirtualQOS*** interface.

Figure 8 shows the M-, D- and C- planes of the XRM and the respective connection management related objects (see also Figures 1, 2 and 3) that reside on them. As indicated in the figure, algorithms that provide call and connection level abstractions are built on top of the "local standardized interfaces" lying on the D-plane of the IRM.

In this example, there is a request for a new connection to be setup between switches A and B. When the request is received by the Connection Manager (the client) on the C-Plane, several actions are performed. At first, the Connection Manager will request from the ***Route*** object (on the D-plane) a route from switch A to B. After the Connection Manager has obtained the route, it uses the information to poll each intermediate node's ***Virtual Link*** with a link access request. Based on the information provided by the schedulable region associated with the given link, the link admission control method determines whether the call will go through or not. Once the Connection Manager is assured that the call can be accepted by all links along the route, it calls the interface of the ***Virtual Switch***es and/or ***Virtual Link***s representing the resources along the selected route, requesting a connection to be setup. (For fast call set up this step can be combined with the previous ones.) Because the ***Virtual Switch***es and ***Virtual Link***s contain abstract representations of real physical objects such as switch controllers, virtual path controllers, virtual circuit controllers and link controllers (not shown for lack of space), they can setup the physical connection by simply modifying the values of some of these objects (in this case, to set up the virtual channel, the connection manager invokes the

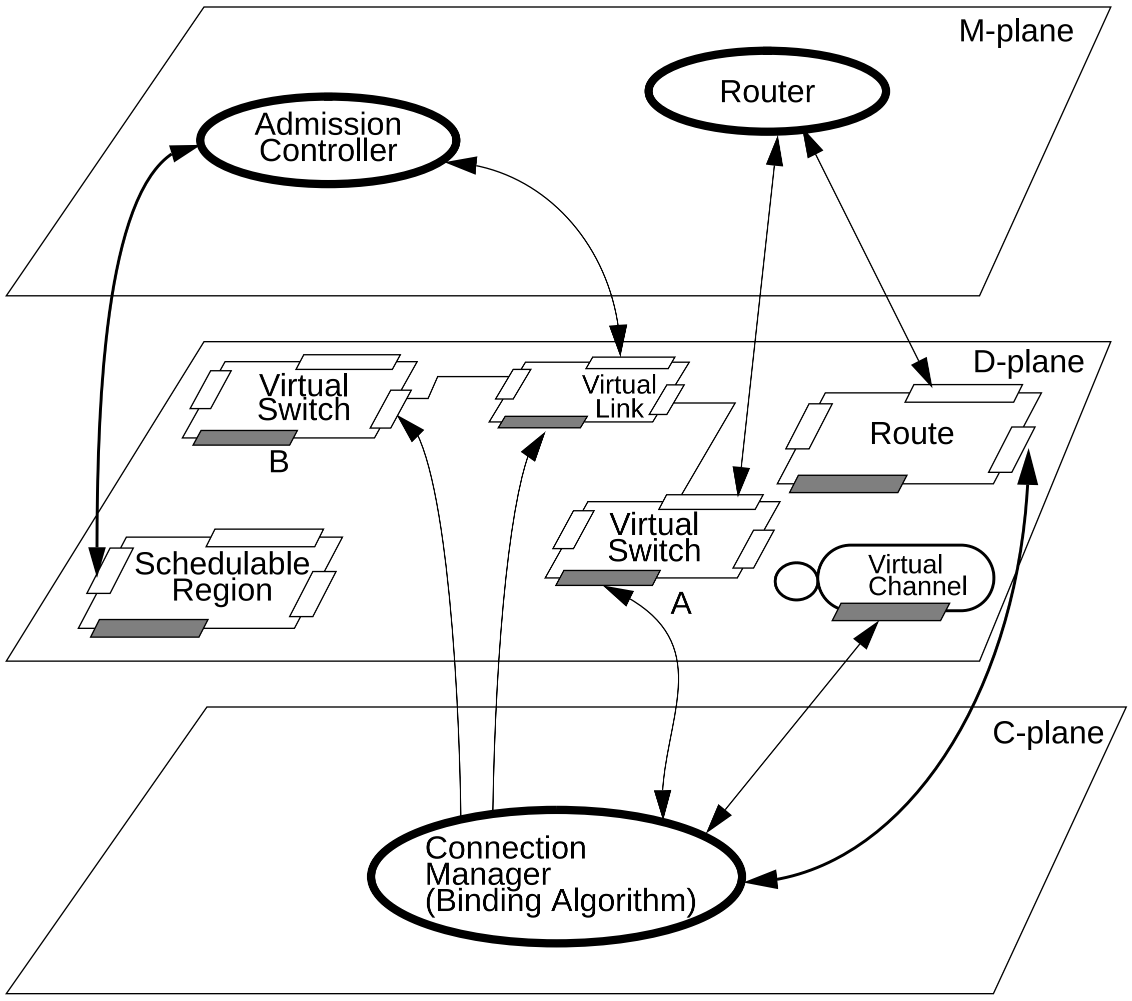

Figure 8. Binding for Connection Management with QOS Guarantees

appropriate method of the ***Virtual Switch*** and ***Virtual Link*** object representing the corresponding physical switches and links between points A and B).

At this stage, the call setup is complete and the Connection Manager's role is over. Whenever a new call is accepted into the system, the operating points of the ***Virtual Switche***s and ***Virtual Link***s along the path of the call changes. After a call disconnect request the connection manager executes the reverse operations.

Once again, note that the example above merely outlines the implementation of a possible connection management algorithm. Other schemes supporting, for example, the Q.93b signalling interface can similarly be implemented using this architecture. A general methodology for designing binding applications is given in [27].

6.2 Parallel Virtual Machine

In this section, we further illustrate the generality of our architecture with an example of computational binding. Again, the emphasis is not on the application itself, but on the fact that even generalized distributed computing tools can be easily built on top of our architecture. The Parallel Virtual Machine is an example of a binding application built upon our binding architecture.

The Parallel Virtual Machine (PVM) [13] is a public domain distributed computing software library developed at the Oak Ridge National Laboratory for facilitating the development of general purpose distributed computing applications. Essentially, PVM presents a reliable connectionless data service interface to applications, thereby freeing them from the concerns of the underlying network. PVM also provides a registration facility that allows applications to register themselves with a specified name. This provides flexibility for reconfiguration because applications are identified only through their registered names and not by their locations (such as a network address). Note that the binding between a name and its associated instance is static, i.e., registration is performed only once at application start-up time and cannot be subsequently changed. Several other useful service primitives are also provided for the convenience of distributed system developers. These include facilities for synchronizing parallel applications, primitives for configuration management, process control and group management.

Figure 9 shows the interface inheritance tree for PVM. A ***PVMService*** interface generic to all PVM services is defined. Within this, three specific interfaces are further specified. The ***SynchronizationService*** defines an interface of the object providing synchronization to PVM clients. Clients that wish to be synchronized send requests to the ***SynchronizationService*** object and wait for a reply. When the last client has sent the request, the ***SynchronizationService*** signals to all the waiting clients. The ***ConfigurationService*** object allows clients to obtain information about the system configuration (e.g., what other clients are running). Finally, the ***ProcessManagementService*** object allows PVM processes to manipulate other PVM processes. For example, a client may want to spawn off child processes to perform some task and destroy them after that.

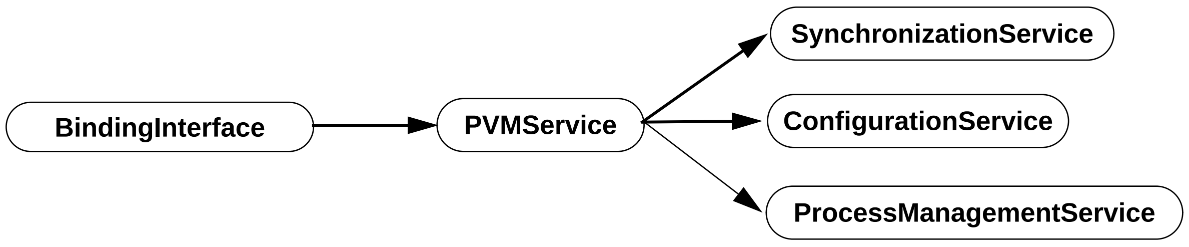

Figure 9. Interface Inheritance for PVM

Figure 10 shows the implementation of clients, synchronization, process control and configuration management services as objects. Note that PVMs registration service is now not explicitly required because CORBA automatically provides a naming service to all objects. This allows greater flexibility than the PVM name registration because the association is not static and objects may be freely moved. As each client is now an object, special primitives for passing data between themselves are no longer required. Client objects can simply call each other supported by the respective standard interfaces. In a similar manner, one or more client objects can call on a service object for the desired service to be performed. In Figure 10, suppose Client A wants to synchronize with Client B. A first informs the ***Synchronization Service*** object and then goes to sleep. When B calls the ***Synchronization Service*** object, A is notified. Similarly, when Client A wants to spawn off a child process, Client C, it requests the ***ProcessManagementService*** to execute an UNIX 'remote shell' command to start up the child process at an appropriate network node.

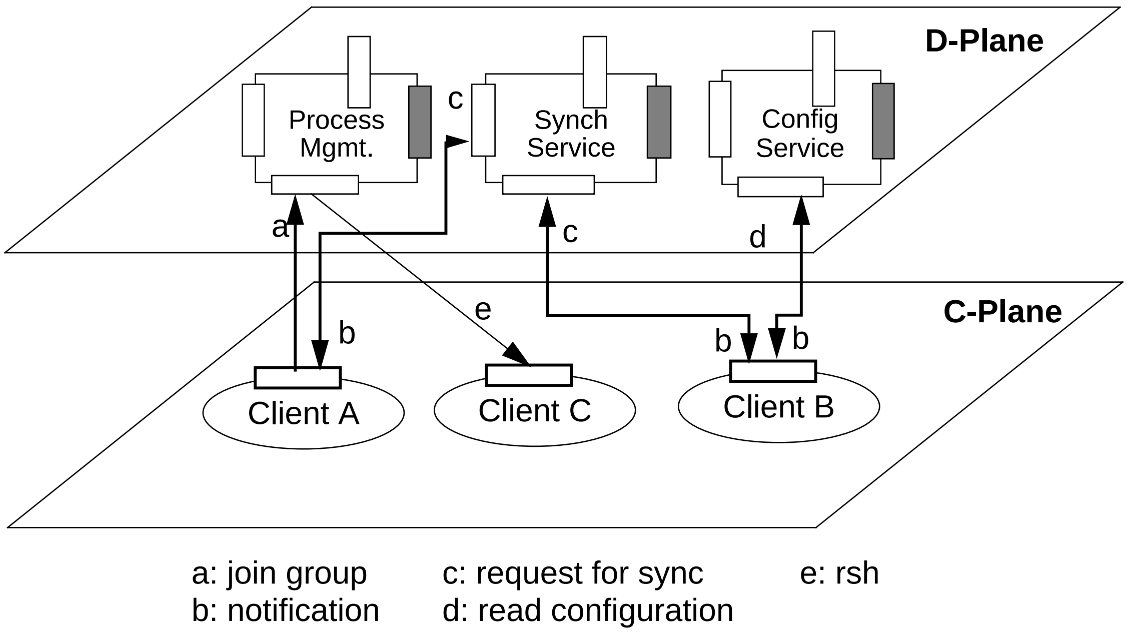

Figure 10. PVM as an example of Computational Binding

PVM is an example of what we call computational binding. It can be implemented as shown directly above our binding architecture. The various interfaces that implement binding operations in PVM, can be derived directly from the BIB interfaces of the binding architecture. PVM will require certain C- and M- plane specific binding algorithms. The implementer has the flexibility to implement these above the binding interface base.

7. Conclusion and Future Directions

The simplicity of our binding architecture is intentional. The idea was to present the concept of binding in the simplest possible way and illustrate the advantages. In the future more sophisticated BIB interfaces can be added for modeling new broadband networking devices and concepts. The design of the binding methods and primitives is also evolutionary and we have only described a bare minimum of these.

We have intentionally left out the description of many “object management” related primitives like interface instantiation and deletion, interface location and interface migration, etc. These are standard activities that many object based systems provide and we have been emphasizing that by using CORBA, we have comfortably bypassed many of these issues and concentrated mainly on multimedia networking requirements.

The advantages of our binding architecture are manifold. Firstly, by providing open interfaces such an architecture ‘‘naturally’’ satisfies the requirements for defining multimedia services both within the network and the multimedia computing platforms. Secondly, our architecture facilitates guaranteeing end-to-end QOS as it seamlessly supports cooperation among distributed algorithms. Thirdly, it provides efficient ways of supporting distributed binding algorithms. For example, a connection establishment algorithm could be carried out either sequentially or in parallel.

Our binding architecture, defined by the BIB and binding algorithms, is conceptually similar to the MIB and CMIS/CMIP protocol of the OSI management architecture. Hence, the integration of the control architecture with the management architecture of multimedia networks becomes greatly simplified. By putting the BIB and the MIB into the Telebase (D-plane), the sharing of data among the C-, M- and N-planes becomes manageable. Using IDL and GDMO for representing information in the BIB and MIB, respectively, further simplifies this task.

As expected, many issues still remain open or have been simply left out because of space limitations. Currently, an implementation for validating our design is underway.

References

[1] Andrews, R.G., “Paradigms for Process Interaction in Distributed Programs”, *ACM Computing Surveys*, Vol. 23, No. 1, March1991, pp. 49-90.

[2] ANSAware Version 4.1 Manual, Architecture Projects Management Ltd., Cambridge, UK, May 1992.

[3] Appeldorn, M., Kung, R. and Sarraco, R., “TMN + IN = TINA”, *IEEE Communications Magazine*, March 1993, pp. 78-85.

[4] Barr, W.J., Boyd, T. and Inoue, Y., “The TINA Initiative”, *IEEE Communications Magazine*, March 1993, pp. 70-76.

[5] Bellcore Information Networking Research Laboratory, “Touring Machine System”, CACM, Vol. 36, No. 1, Jan. 1993, pp. 68-77.

[6] Black, U., Network Management Standards, McGraw-Hill Inc., New York, NY, 1992.

[7] CCITT: Recommendation I.413, “B-ISDN User Network Interface”, Geneva, 1991.

[8] CCITT: Recommendation Q.761, “Functional Description of the ISDN User Part of Signalling System No. 7,” Blue Book, Fascicle VI.8, Geneva, 1989.

[9] Chin, R.S. and Chanson, S.T., “Distributed Object-Based Programming Systems”, *ACM Computing Surveys*, Vol. 23, No. 1, March1991, pp. 91-124.

[10] Common Object Request Broker: Architecture and Specification. OMG document 91-12-1.

[11] Crutcher, A. L. and Waters, A. G., “Connection Management for an ATM Network”, *IEEE Network*, November 1992, pp. 42-55.

[12] Gaddis, M.E., Bubenick, R. and J.D. DeHart, “A Call Model for Multipoint Communication in Switched Networks”, Proceedings of the *International Conference on Communications*, 1992.

[13] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V., PVM 3.0 User’s Guide and Reference Manual, Oak Ridge National Laboratory, Tennessee, February 1993.

[14] Heijenk, G. J., Hou, X. and Niemegeers, I. G., “Communication Systems Supporting Multimedia Multi-user Applications”, IEEE Network, Vol. 8, No. 1, Jan/Feb 1994, pp. 34- 44.

[15] HP, IBM and SunSoft, “Multimedia Systems Services Architecture”, Response to the *Multimedia System Services Request for Technology of IMA*, June 1, 1993.

[16] Hyman, J.M., Lazar, A.A. and Pacifici, G., “VC, VP and VN Resource Assignment Strategies for Broadband Networks”, Proceedings of the *Workshop on Network and Operating Systems Support for Digital Audio and Video*, Lancaster, United Kingdom, November 3-5, 1993, pp. 99-110.

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

[18] Lazar, A.A., “A Real-Time Control, Management and Information Transport Architecture for Broadband Networks”, Proceedings of the *1992 International Zurich Seminar on Digital Communications*, March 16-19, 1992, pp. 281-296.

[19] Lazar, A. A., “Challenges in Multimedia Networking”, Proceedings of the *International Hi-Tech Forum*, Osaka, Japan, February 24-25, 1994, pp. 24-33.

[20] Lazar, A.A., “A Research Agenda for Multimedia Networking”, Proceedings of the *Workshop on Fundamentals and Perspectives on Multimedia Systems*, International Conference Center for Computer Science, Dagstuhl Castle, Germany, July 4-8, 1994.

[21] Lazar, A. A., Bhonsle S., Lim, K. S., “Specification of the Binding Architecture, Version 1.0”, July 1994.

[22] Lazar, A.A., Ngoh, L.H. and Sahai, A., “Multimedia Networking Abstractions with QOS Guarantees”, *Technical Report # 375-94-22*, Center for Telecommunications Research, Columbia University, New York, NY 10027-6699, August 1994.

[23] Lazar, A.A. and Pacifici. G., “Control of Resources in Broadband Networks with Quality of Service Guarantees”, *IEEE Communications Magazine*, Vol. 29, No. 10, October 1991, pp. 66-73.

[24] Lazar, A.A. and Stadler, R., “On Reducing the Complexity of Management and Control of Future Broadband Networks”, Proceedings of the *Workshop on Distributed Systems: Operations and Management*, Long Branch, NJ, October 4-6, 1993.

[25] Leopold, H., Coulson, G., Frimpong-Ansah, K., Hutchison, D. and Singer, N., “The Evolving Relationship between OSI and ODP in the New Communications Environment”, *Presented at 2nd RACE International Conference on Broadband Islands*, Athens, Greece, June 15-16, 1993.

[26] Minzer, S., “A Signalling Protocol for Complex Multimedia Services”, *IEEE Journal of Selected Areas in Communications*, Vol. 9, No. 9, Dec. 1991, pp. 1383-1394.

[27] Pacifici,G. and Stadler, R., “Separating Policy Implementation from Policy Execution: A Paradigm for Resource Management in Broadband Networks”, Proceedings of the Network Operations and Management Symposium, Kissimmee, FL, February 1994.

[28] Partridge, C. and Pink, S., “An Implementation of the Revised Internet Stream Protocol (ST-II)”, *Journal of Internetworking: Research and Experience*, Vol. 3, No. 1, March 1992.

[29] Zhang, L., Deering, S., Estrin, D., Shenker, S. and Zappala, D., “RSVP: A New Resource ReSerVation Protocol”, *IEEE Networks Magazine*, September 1993.