Integrating Multimedia Streams into a Distributed Computing System

BJ Murphy

Computer Laboratory, University of Cambridge, UK

GE Mapp

Olivetti Research Laboratory, Cambridge, UK

ABSTRACT

Continuous media, such as audio and video, are quickly becoming an integral part of distributed computing environments. A shortcoming of such environments is their lack of support for continuous flows of information. What is missing is the notion of an on-going communication activity with an associated quality of service.

This paper describes a model for integrating multimedia flows into a distributed computing system. The model permits explicit bindings to be established between type-checked stream interfaces. The stream binding is represented in the computational model as a first-class object which encapsulates configuration rules and QoS attributes. An operational interface supplied by the binding object allows other objects within the system to manage the binding, to renegotiate QoS parameters, to control the flows across the binding, and to register interest in stream events such as flow reports and communication errors. The in-band stream interface is an abstract C++ wrapper around transport mechanisms that include intra-host IPC and network transport protocols such as TCP and XTP.

A prototype implementation of this model is described using the Common Object Request Broker Architecture (CORBA). The implementation environment comprises a local area ATM network with directly attached multimedia peripherals and general purpose workstations.

Keywords: Continuous Media, Streams, CORBA, ODP.

1 INTRODUCTION

Continuous media, such as audio and video, are becoming an integral part of distributed computing environments. This trend is being accelerated by the adoption of the "exploded workstation" model by various research groups.[6,14] In this model, multimedia peripherals are attached directly to the network. Distributed computing techniques are required to provide the user with a coherent view of networked resources.

Distributed systems are often modelled using an object-oriented approach. An object encapsulates state and provides a well-defined interface to the rest of the system. Objects interact by invoking operations on these interfaces. An object-oriented computational model of a distributed system maps conveniently onto an engineering implementation in which invocations are performed by Remote Procedure Call (RPC).

A common shortcoming of such models is their failure to address *continuous* flows of media such as audio and video. The problem is that the transfer of continuous media cannot be adequately expressed in terms of invocation. What is missing is the notion of an *on-going* communication activity with an associated "quality of service" (QoS).

This paper describes a computational model for distributed systems in which continuous media flows are established and controlled by "stream" binding objects. The term "binding" is used here in a very general sense to refer to a logical communication channel between two or more objects which may include manipulation of data on that channel. The basic notion of a stream is a uni-directional flow of a series of messages of a pre-defined type. More complex abstractions can be derived from the basic stream interface. Such "smart streams" can be used, for example, to simplify the management of inter-related flows and to provide "in-band" processing of stream data. The model presented here is briefly set in context with other work in this field. An implementation of this model based on the Common Object Request Broker Architecture (CORBA)[11] is presented.

2 COMPUTATIONAL MODEL

We refer to objects that handle multimedia flows as "devices". Devices are further sub-classed as either "sources", "sinks" or "modules". A source is a media producer and is normally an abstraction of a media-generating hardware device such as a camera or microphone. A sink is a media consumer and is normally an abstraction of a media-rendering hardware device such as framebuffer or loudspeaker. A module is a media processor that accepts incoming data, processes that data in some way, and outputs the modified data. Devices are "virtual" entities in the sense that there may be multiple logical devices per physical device. Devices interact via communication end-point interfaces called "ports". A device supplies an operational (RPC) control interface by which it may be managed.

We refer to objects that establish and control multimedia flows as "stream" binding objects. Figure 1 shows a number of possible stream bindings between devices. In the simplest case, a stream represents a single (unicast) flow of data from producer to consumer (Figure 1a). A more general type of stream is a multicast flow from a single producer to multiple consumers (Figure 1b). Both types of binding are characterised by the fact that no manipulation of the flow is performed as the data moves from producer to consumer. We describe such bindings as "passive" bindings.

An additional class of binding involves the processing of data between producer and consumer. For example, Figure 1c depicts a stream binding object that performs pattern recognition on an incoming video stream and generates an outgoing stream of co-ordinates that represents the position of, say, a hand or a face. Figure 1d represents a stream binding object that selects one of a number of audio streams depending on input volume level. We refer to such bindings as "active" bindings. An active binding is the composition of two or more passive bindings separated by one or more processing modules such as pattern recognisers or threshold detectors. Where possible, we wish to insulate application writers from the complexity of constructing and configuring the internals of such composite bindings by providing a "repository" of commonly used active bindings.

An additional style of composition is one in which multiple parallel flows need to be co-ordinated, often because there are synchronisation constraints between them. The classic example is the maintenance of lip synchronisation when the audio and video flows each have their own network connection and are handled by separate end-point devices. We collectively refer to the various styles of active binding and the various styles of binding composition as "smart streams".

The stream binding object provides an operational (RPC) stream control interface to the rest of the system.

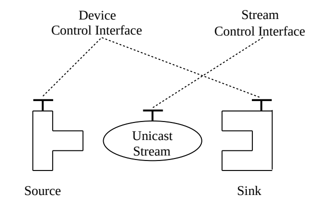

Figure 1a: Unicast Passive Stream

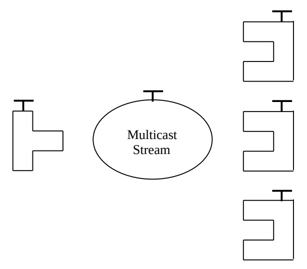

Figure 1b: Multicast Passive Stream

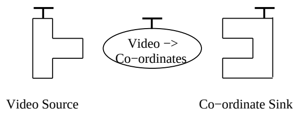

Figure 1c: Active Stream: Video Pattern Recogniser

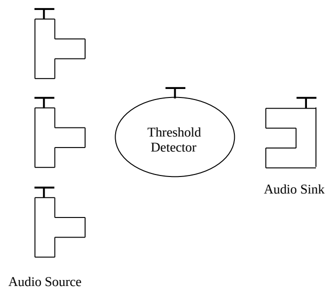

Figure 1d: Active Stream: Audio Switch

Figure 1: Examples of Stream Bindings in the Computational Model

The purposes of this interface are:

- to manage the binding (eg delete the binding, add members to a multicast binding)
- to manage QoS parameters (eg to permit dynamic QoS re-negotiation)
- to control the flow of data (eg start, stop, pause)
- to register callbacks for binding-related events (eg QoS degradation events, multicast member change events)

A stream control interface is typed in terms of the configuration of the binding (eg unicast or multicast), the processing, if any, performed by the binding (eg mixing or level detection), and the nature of the flows involved (eg audio or video).

The stream control interface encapsulates the underlying signalling mechanism that is used to manage the binding. This mechanism may be based on the conventional end-to-end exchange of signalling messages (such as that defined by the ATM Forum for the management of ATM virtual connections[1]) or on other techniques such as the use of CORBA to provide direct control of network components.[9]

Associated with the basic stream interface is a set of QoS attributes that governs the flow. These attributes include:

- bandwidth (average, peak, burst interval, tolerance)
- delay (maximum, tolerance)

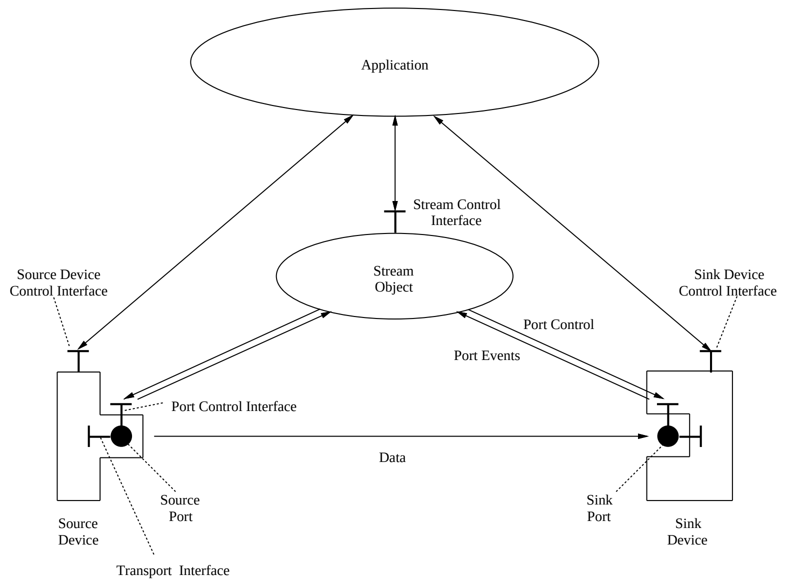

Figure 2: The Engineering Model of a Unicast Stream

- jitter (maximum, tolerance)
- reliability (reliable or unreliable in terms of error-free delivery)
- cost (maximum, tolerance)

The stream binding object provides QoS mapping functions between type-specific QoS specifications (such as video frame-rate, image pixel-depth) and transport-specific QoS parameters (such as delay and bandwidth). A smart stream needs to map application-specific notions of QoS to transport-specific QoS parameters for each passive binding and to processing-specific QoS parameters for each module. For example, a smart stream that converts audio to text may need to map a parameter such as "word spotting accuracy" onto underlying communication and processing resource requirements. In order to manage *end-to-end* QoS,[2] the application, or some higher-level QoS manager, needs to co-ordinate operations at the stream interface and at the end-point device interfaces.

3 ENGINEERING MODEL

Operational interfaces defined in the computational model are specified using CORBA Interface Description Language (IDL). All control interactions are (implicitly) accomplished using CORBA RPC communications. Data transfer, on the other hand, is achieved using Inter-Process Communication (IPC) mechanisms better suited to the transfer of continuous media and for which QoS control can be provided. This is discussed further below.

Figure 2 shows the engineering of a unicast stream binding. There is a one-to-one mapping between (virtual) devices in the computational model and CORBA objects in the engineering model. Communication end-points (ports) which are opaque in the computational model become visible in the engineering model. A port offers two interfaces to the rest of the system: a local (ie non-IDL) "transport" interface for data transfer and an IDL

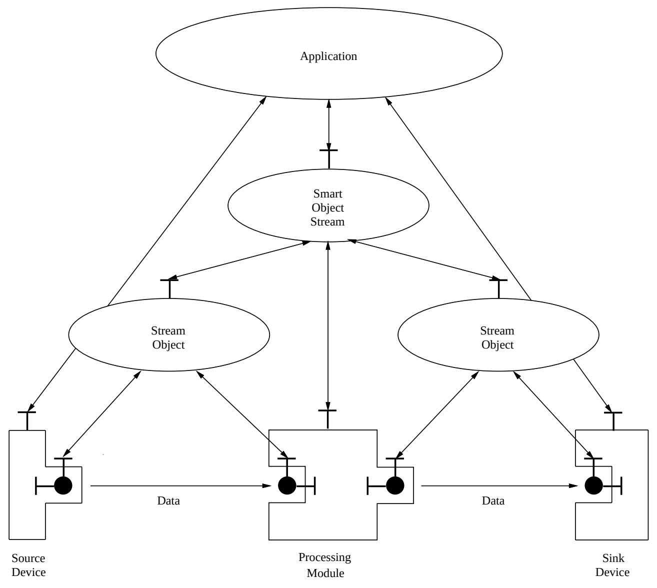

Figure 3: The Engineering Model of a Simple Smart Stream Comprising a Single Processing Module

port control interface. The port is typed in terms of the nature of the messages supported by the flow (eg audio samples, video frames) and the direction (ie in or out) of the flow.

The stream object abstracts over the collection of ports that are involved in the underlying stream data flow. There is a one-to-one mapping between passive bindings in the computational model and stream objects in the engineering model. Smart streams, on the other hand, are engineered as hierarchies of stream objects and processing modules (see Figure 3). The stream (or "super-stream") object dispatches control operations invoked on the stream control interface to the various ports (or "sub-streams" and modules) that are involved in the stream data flow. The stream object may arrange for the insertion of presentation-level transformation objects between ports in order to solve format and coding incompatibilities.

4 RELATIONSHIP TO OTHER ARCHITECTURES

A number of architectures have recognised that communication requirements and programming paradigms for continuous media are different to those for control interactions. The ISO Open Distributed Processing Reference Model (ODP-RM)[8] defines a "stream interface" to compliment the conventional semantics of an operational interface. The "signature" of a stream interface defines the set of flows supported at that interface, and, for each flow, the role of the interface (producer or consumer). Furthermore, an "environment", which may include QoS requirements, is associated with a computational interface and is used to influence *explicit* bindings between these interfaces. The ODP philosophy on streams has been adopted by the Telecommunications Information Networking Architecture Consortium (TINA-C).[5]

Much of the thinking behind the ODP-RM has been derived from work on the Advanced Network Systems Architecture (ANSA).[12] The original ANSA architecture did not address the requirements of multimedia. A number of enhancements were suggested to redress this deficiency: researchers from the University of Lancaster proposed a set of "multimedia base services" within the context of the existing computational model[4] while Nicolaou suggested changes to the computational model itself.[10] The current ANSA Phase III Distributed Interactive Multimedia Architecture (DIMMA) project is pursuing the latter approach as well as addressing the problems of interworking between various engineering implementations.

A consortium of computer vendors has proposed the Multimedia Systems Services (MSS) architecture[7] in response to a call for technology from the Interactive Multimedia Association (IMA). The CORBA-based MSS architecture defines a standard set of services that can be used by multimedia application developers in a heterogeneous distributed computing environment. CORBA is also the basis for a proposal from the COMET group at the University of Columbia for a global binding architecture that integrates network and end-system resources.[9]

Our architecture has more in common with the MSS and Lancaster approaches than with the ODP-related approaches. Both the MSS and Lancaster models have the concept of a "stream" (called a "VirtualConnection" in the MSS model) which provides an interface to the underlying communication channel (which may be unicast or multicast) and permits *dynamic* control of that channel. The ODP-related models recognise the need for an explicit binding between a pair of compatible computational interfaces, each with an associated set of statically defined QoS attributes, but they do not provide a mechanism for dynamic QoS changes to that binding. A multi-party binding is modelled either as a "compound" binding in which a "binding object" sits in the data path and maintains "primitive" bindings to each endpoint or as a static multicast binding with no interface for making dynamic changes to group membership.

We differ from the MSS and Lancaster models to the extent that we believe that the stream object should be typed and should encapsulate type-specific QoS mapping functions. Furthermore, we believe that, to some degree, communication endpoints (which we call "ports") should be statically typed in order to detect gross configuration errors (such as an attempt to connect a video source to an audio sink) at compile time instead of at bind time. We are currently contemplating the degree to which ports (rather than messages) can be typed without restricting scope for polymorphic processing modules, without eliminating the possibility for in-band stream annotations, and without pushing technology dependencies into the typing system.

In common with the MSS and Lancaster models, the data transfer interface (which we call the "transport" interface) is not specified in a computational language such as CORBA IDL and the data transfer mechanisms (such as those based on network protocols) fall outside the scope of the computational model. For architectural consistency, we endorse efforts to add a stream interface to the computational model and a QoS specification to each computational interface. For pragmatic reasons, we are stepping outside the computational model for the transfer of continuous media although we are making *control* of these flows possible via CORBA IDL interfaces.

5 IMPLEMENTATION

A prototype implementation of parts of the engineering model has been performed. The environment in which we are working comprises a local ATM network to which a variety of multimedia peripherals (eg "video bricks", "audio bricks", "video tiles") and general purpose workstations are attached. Each multimedia peripheral is directly attached to the network resulting in a highly distributed architecture. The ATM network includes both wired and wireless segments; the impact of mobility on streams is of great interest to us.

The general purpose workstations (running Windows NT and various flavours of UNIX) and the multimedia peripherals (running an in-house microkernel called ATMos) all support an implementation of CORBA: the vendor platforms run Orbix from IONA Technologies and the ATMos machines run an in-house Orbix-compatible ORB called omniORB. All platforms support the TCP/IP protocol stack over ATM/AAL5. In addition, the UNIX and ATMos platforms support an implementation of XTP[3] over IP/AAL5 and "native" AAL5. We regard XTP as more suitable for the transfer of continuous media than conventional transport protocols such as TCP primarily because it can be configured to provide a variety of transport services. XTP supports byte-stream or messageoriented transfer semantics, optional error detection and correction, timer-based rate control in conjunction with window-based flow control, and reliable multicast in the local area.

In our streams implementation, every node runs one or more object "factories" (ie processes that can dynamically create CORBA objects). On ATMos, these are configured as CORBA "persistent" servers; on other platforms, factories may be launched dynamically by a daemon process. In order to establish a stream channel, the application asks a (typically local) factory to create a stream object of the specified type and with a specified set of QoS attributes. At present, we only support passive stream bindings involving primitive flow types (audio and video) with no QoS support other than the choice of transport protocol. The application then asks the factories on the source and sink nodes to create devices of the appropriate type. For example, an application may ask the source factory to create a "Camera" device and the sink factory to create an "XVideoWindow" device. The resulting object references allow the application to control the devices. For example, a Camera object might support a variety of picture sizes and frame rates. Note that operations that affect *all* devices within a given node, for example panning or zooming a camera on a video brick, are part of the node-specific factory interface rather than the virtual device interface. That is, the device interface encapsulates the resources of a node that may be controlled independently of other devices within the node.

The device object reference allows the application to obtain (or create) a port of an appropriate "polarity" (ie in or out). The application may then invoke operations on the stream control interface to "attach" a source port and one or more sink ports to the stream. The type of the stream object determines the permitted configuration of sources and sinks. When an appropriate number of ports have been attached, the stream may be instructed to create a stream channel between the group of ports. The mechanism by which this is achieved is entirely outside CORBA. At present, the stream channel uses either the XTP or TCP transport protocols depending on the QoS attributes of the stream. The intention is that different transport protocols can be inserted without impacting applications.

The port transport interface is implemented as a C++ abstract class that provides a generic data transfer interface to device programmers. As such, it is similar in spirit to Washington University's ACE wrappers.[13]

The port control interface is an IDL interface that, in addition to QoS control, provides a generic binding capability based on an abstract representation of a transport interface address. Transport interfaces are bound together by port control operations, invoked by the stream object, of the form `port1->import(port2->export())`.

6 FUTURE WORK

Work is required to add QoS support to the implementation. In particular, the semantics of and mechanisms for QoS renegotiation need to be understood. For example, signalling services for connection-oriented network technologies such as ATM may not permit in-call QoS renegotiation. This may require the stream object to tear down and re-establish a transport connection during the lifetime of the stream. Furthermore, end-to-end QoS guarantees imply the need for management of end-system resources such as memory and CPU cycles as well as communication resources such as bandwidth.

Additional work is required to support the concept of smart streams. We are particularly interested in those streams that perform on-the-fly transformations of media flows. Experience from previous projects[14] has suggested that streams should be allowed to flow through multiple, perhaps tens, of modules between source and sink. The above architecture makes the tacit assumption that most communication between devices is across a network. In the case of "stream pipes", this will not be true. What is needed are efficient ways of passing multimedia data between devices within an address space and between address spaces but within a single host (for which no presentation-layer conversions are required). To this end, we are investigating buffer representations, buffer editing primitives, and IPC mechanisms that minimise copying and unnecessary processing.

7 CONCLUSIONS

Just as we regard best-effort operating systems as inadequate for multimedia, so we regard best effort communication mechanisms as inadequate for continuous media. The implication is that interfaces are required to allow explicit manipulation of processing and communication resources. Furthermore, since applications that deal with multimedia are typically long-lived, we think it necessary to be able to control these resources dynamically (ie during the lifetime of the application). We believe that our stream control interface gives us a “handle” on the underlying communication resource. Furthermore, the stream control interface abstracts away from details of the signalling mechanism while the transport interface abstracts away from details of the data transfer mechanism; this gives us flexibility in the implementation of these mechanisms. We await implementation and operational experience before being able to assess the benefits of our architecture.

8 REFERENCES

[1] ATM Forum, *ATM User-Network Interface Specification*, Version 3.0, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] A Campbell, G Coulson, D Hutchison, “A Quality of Service Architecture”, *ACM SIGCOMM, Computer Communications Review*, June 1994.

[3] G Chesson, “XTP/PE Design Considerations”, *Proceedings of 13th Conference on Local Computer Networks*, October 1988.

[4] G Coulson, GS Blair, N Davies, N Williams, “Extensions to ANSA for multimedia computing”, *Computer Networks and ISDN Systems*, No 25, 1992.

[5] A Van Halteren, P Leydekkers, H Korte, “Specification and Realisation of Stream Interfaces for the TINA-DPE”, *Proceedings of TINA’95 Conference*, Melbourne, Australia, February 1995.

[6] H Houh, J Adam, M Ismert, C Lindblad, and DL Tennenhouse, “The VuNet Desk Area Network: Architecture, Implementation, and Experience”, *IEEE Journal of Selected Areas in Communications*, Vol 13, No 4, May 1995.

[7] HP, IBM and Sun, *Multimedia System Services*, a response to the Multimedia System Services Request for Technology from the Interactive Multimedia Association, June 1993.

[8] International Standards Organisation, *Basic Reference Model of Open Distributed Processing, Part 3: Architecture*, ITU/T X.903-ISO 107046-3, February 1995.

[9] AA Lazar, S Bhonsle, KS Lim, “A Binding Architecture for Multimedia Networks”, *Proceedings of the Multimedia Transport and Teleservices*, Vienna, Austria, November 14-15, 1994.

[10] C Nicolaou, “An Architecture for Real-Time Multimedia Communication Systems”, *IEEE Journal of Selected Areas in Communications*, Vol 8, No 3, April 1990.

[11] Object Management Group, *Common Object Request Broker Architecture and Specification*, OMG document 91-12-1, available electronically via `http://www.omg.org`, 1991.

[12] O Rees, *The ANSA Computational Model*, APM document APM.1001.01, available electronically via `http://www.ansa.co.uk/ANSA`, January 1994.

[13] D Schmidt, T Harrison, E Al-Shaer, “Object-Oriented Components for High-speed Network Programming”, *USENIX Conference on Object-Oriented Technologies*, Monterey, CA, June 1995.

[14] S Wray, T Glauert, A Hopper, “The Medusa Applications Environment”, *Proceedings of International Conference on Multimedia Computing and Systems*, Boston, MA, USA, May 1994.