

## **A Software Architecture for Adaptive Distributed Multimedia Systems**

Tom Fitzpatrick, Gordon S. Blair, Geoff Coulson, Nigel Davies and Philippe Robin  
Distributed Multimedia Research Group,  
Department of Computing,  
Lancaster University,  
Lancaster LA1 4YR,  
U.K.

E-mail: [tf, gordon, geoff, nigel, pr]@comp.lancs.ac.uk

## **Abstract**

In order to support multimedia applications in mobile environments, it will be necessary for applications to be aware of the underlying network conditions and also to be able to adapt their behaviour and that of the underlying platform. This paper focuses on the role of middleware in supporting such adaptation. In particular, we investigate the role of open implementation and reflection in the design of middleware platforms such as CORBA. The paper initially extends CORBA with the concept of explicit binding, where path of communication between objects is represented as first class objects. We then introduce the concept of open bindings which support inspection and adaptation of the path of communications. An implementation of open bindings for adaptive continuous-media interaction is described using the example of adaptive video-on-demand for mobile environments.

## 1. Introduction

Future computer systems will consist of end-systems which will be either disconnected, weakly connected by low speed wireless networks such as GSM, or fully connected by fixed networks ranging from Ethernet to ATM. Furthermore, the level of connectivity will vary over time as a consequence of the mobility of the modern computer user. Even when connected to a particular network, fluctuations in throughput and delay may be experienced due to congestion, e.g. as witnessed in the Internet.

To cope with such variations, it is important that systems can adapt to the quality of service (QoS) offered by the network. Such adaptation can take place at a variety of levels in the system, e.g. in the communications stack, in the operating system or in the application. In this paper, we are concerned with support for adaptation in the middleware platform, i.e. the layer of software above the operating system, which offers a platform independent programming model and hides problems of heterogeneity and distribution. More specifically we consider the design of middleware platforms which (i) allow the application to inspect the current level of QoS at various points of the system, and (ii) enable applications to dynamically adapt their behaviour or the behaviour of the underlying platform in response to changes in QoS. We are particularly interested in adaptation as required by multimedia applications.

A range of middleware technologies is now available, including CORBA, DCE, and DCOM. In addition, ISO have recently completed an international standard defining a Reference Model for Open Distributed Processing (RM- ODP); this standard provides a framework for the development of middleware platforms. In this paper, we focus on the CORBA platform from OMG, although many of the arguments could be applied to other platforms. CORBA provides an environment whereby objects can interact in a distributed environment. Objects are defined in a language and platform independent manner through an Interface Definition Language (IDL). An Object Request Broker enables clients to issue requests on an object; the ORB locates the object, transmits the request, prepares the object implementation for receiving and processing the request, and conveys results back to the client. (Further details of CORBA can be found in [Blair98a].)

A major problem with CORBA is that the architecture adopts a traditional black box approach whereby the implementation of the platform is hidden from the application. Until recently, this has not been a great problem. Indeed, it could be argued that this is a highly desirable property of a middleware platform. With the advent of mobile (multimedia) computing, however, such an approach is untenable; in such environments, it is essential to have

(selective) access to the underlying implementation. To achieve this, we adopt concepts from open implementation [Kiczales91] and reflection [Maes87]. In this paper, we focus on the use of such techniques to enable inspection and adaptation of the path of communication between interacting objects. In other research at Lancaster, we also consider the use of such techniques in other aspects of a middleware platform (e.g. concurrency control, thread scheduling and real-time synchronization) [Blair98b]

The paper is structured as follows. Section 2 presents some background material on open implementation and reflection. Section 3 then considers our approach to supporting multimedia in CORBA, focusing on the concept of explicit bindings. Following this, section 4 presents an extension to explicit bindings, referred to as open bindings. A short example is presented in section 5, after which the role of open bindings in supporting inspection and adaptation is then considered in section 6. Our current implementation of open bindings is discussed briefly in section 7, while Section 8 examines some related work. Finally, section 9 presents some concluding remarks.

## **2. Background on Open Implementation and Reflection**

The concept of open implementations has recently been investigated by a number of researchers, most notably Kiczales et al at Xerox PARC [Kiczales91]. The goal of this work is to overcome the limitations of the black box approach to software engineering and to open up key aspects of the implementation to the application. This must however be achieved in such a way that there should be a principled division between the functionality they provide and the underlying implementation. The former can be thought of as the base interface of a module and the latter as a meta-interface [Rao91].

The role of reflection is then to provide a principled means of achieving open implementation. In a reflective system, the meta-level interface provides operations to manipulate a causally connected self-representation of the underlying implementation. According to Maes [Maes87], a system is said to be causally connected to its domain if "the internal structures and the domain they represent are linked in such a way that if one of them changes, this leads to a corresponding effect on the other". Such a system has the benefits that, firstly, the self representation always provides an accurate representation of the system, and that, secondly, a reflective system can bring modifications or extensions to itself by virtue of its own computation. In other words, a reflective system naturally supports inspection, and adaptation:

### **1. Inspection**

Reflection enables applications to observe the occurrence of arbitrary events in the underlying implementation. Such an approach can be used to implement functions such as QoS monitors or accounting systems in a portable manner.

## 2. Adaptation

Similarly, reflection allows applications to adapt the internal behaviour of the system either by changing the behaviour of an existing service (e.g. tuning the implementation of message passing to operate more optimally over a wireless link), or dynamically reconfiguring the system (e.g. inserting a filter object to reduce the bandwidth requirements of a communications stream). Such steps are often the result of changes detected during inspection (see above).

Most of the early research in reflection focused on the field of programming language design [Kiczales91, Watanabe88]. More recently, the work has diversified with applications of reflection in areas such as windowing systems [Rao91] and operating systems [Yokote92]. There is also growing interest in the use of reflection in distributed systems. Pioneering work in this area was carried out by McAffer [McAffer96]. More recently, researchers at APM have developed reflective extensions to Java with a view to supporting distributed applications. However, there has been much less activity to date in the design of reflective middleware. Campbell at Illinois has carried out initial experiments on reflection in Object Request Brokers (ORBs) [Singhai97]. The level of reflection however is coarse-grained and restricted to invocation, marshalling and dispatching. In addition, the work does not consider key areas such as support continuous media interaction. Manola has carried out work in the design of a "RISC" object model for distributed computing [Manola96], i.e. a minimal object model which can be specialized through reflection. A PhD student of Cointe is also investigating the use of reflection in proxy mechanisms for ORBs [Ledoux97].

## 3. Introducing Explicit Bindings

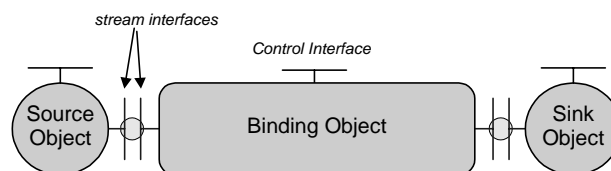
### 3.1. What is Explicit Binding?

In order to address our requirements, it is necessary to extend the programming model offered by CORBA [Blair98a] (thereby aligning it more with ISO RM-ODP). In particular, we introduce the concept of explicit binding. In the current CORBA programming model, binding is implicit in that, when objects interact, an appropriate communications path is created by the underlying ORB. In our approach, we suggest that bindings should be created explicitly by the

programmer; the result of the binding is then an object representing the underlying end-to-end communications path (see Figure 1). One consequence of this approach is that bindings can be created by a third party (in CORBA, the client always initiates an interaction). The importance of explicit binding is twofold:

1. The act of creating a binding can subsume static QoS management functions such as negotiation, admission control and resource reservation.
2. The interface on the binding can be used for dynamic QoS management functions such as inspection and adaptation.

In this paper, we are particularly interested in this second aspect of explicit bindings, i.e. support for inspection and adaptation.



*Figure 1: Explicit Binding*

We introduce two different styles of binding, namely operational bindings and stream bindings: operational bindings support the traditional style of interaction in CORBA, namely operation requests. Stream bindings are then required to support continuous media interaction. A given stream consists of one or more flows where each flow represents the unidirectional transmission of a continuous media type (e.g. audio or video). As a result of this change, it is also necessary to distinguish between operational interfaces and stream interfaces as end-points of operational bindings and stream bindings respectively.

The architecture is open in that bindings are created by an extensible set of binding factories. Factories in CORBA are objects that support the creation of a particular class of object; binding factories are therefore responsible for creating a new binding between a target set of objects. One binding factory could provide the semantics of standard CORBA requests whereas another could provide real-time guarantees in terms of end-to-end latency (perhaps exploiting meta-level functionality to meet the required guarantees). Similarly, for continuous media interactions, one factory could provide a best effort service for video transmission whereas another factory could provide guarantees through an

appropriate resource reservation strategy. In addition, programmers are free to develop their own binding classes, perhaps in terms of existing classes. Explicit bindings provide one step towards a more open architecture in that communication becomes both visible and controllable. This is necessary but, in our view, not sufficient for mobile multimedia applications. We therefore extend this concept further by introducing open bindings.

## **4. The Concept of Open Bindings**

### **4.1. General Approach**

To support mobile computing, it is necessary for the application to be able to exert some control over bindings. One way of achieving this is for binding interface to offer QoS management operations to monitor the current levels of QoS and to adapt to perceived changes. The problem with this approach is that it is very difficult to design a general means of achieving adaptation. This is especially problematic when mobility is introduced due to the proliferation in possible actions. Our approach is for bindings to offer a meta-interface providing access to a causally connected self-representation. This self-representation is provided by an object graph, representing the underlying end-to-end communications path. This equates to a procedural as opposed to a declarative approach to QoS management [Blair97]. We argue that this approach offers the level of flexibility required by mobile computing. This procedural approach is influenced by our experiences with the use of logic or QoS attributes to specify QoS requirements [Coulson96]. We have found that this is a perfectly valid approach for dealing with static QoS properties but the approach cannot easily be extended to deal with adaptation. In Adapt, we still allow the association of some simple QoS attributes defining media types with bindings as a means of checking consistency between interfaces in the bindings (see section 4.3 below). However, the main mechanism for dealing with more dynamic aspects of QoS management is to directly manipulate graphs. Note that this does not preclude the use of declarative techniques which can be built on top of the basic procedural facilities provided by the platform.

### **4.2. Object Graphs**

An object graph consists of processing objects and binding objects which are connected together by *local bindings*. Communication across a local binding is assumed to be instantaneous and reliable, normally implying that local bindings are located in a single address space or a single machine. All other interactions are represented explicitly by the binding objects in the graph. Processing objects then either perform computations on the data flowing through the graph or are

responsible for a particular management function. Examples of processing objects include QoS filters and mixers, QoS monitors, or rate control components. The concept of an open binding is illustrated in Figure 2 below.

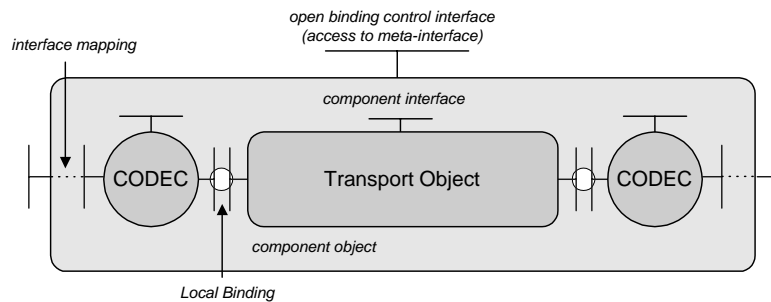


Figure 2: Open Binding

To control visibility of interfaces within a binding, we introduce the concept of interface mapping (illustrated as dotted lines in Figure 2). Interface mapping allows an external interface to map on to the interface of an internal component. The external interface acts as a proxy for the internal interface; all interactions occur at the internal interface via the external interface.

As a further refinement, binding objects can themselves be open bindings and hence also be composed in terms of object graphs. The nesting bottoms out by offering a set of primitive bindings whose implementation is closed. For example, a particular platform might offer RTP or IP services as primitive bindings (depending on the level of openness in the platform). This nested structure provides access to lower levels of the implementation (if required). At a finer granularity, each object in the graph can offer an interface to control its individual behaviour.

### 4.3 Type Checking in Object Graphs

To ensure a basic level of consistency in object graphs, strict type checking is mandatory for all local bindings. When a local binding is requested between two interfaces, a media type must be found which is acceptable to both, otherwise the local binding cannot proceed. This process of media type negotiation requires information on the media type(s) that each interface can produce or consume.

Other researchers have investigated the extending of IDL to include media type description as part of an interface definition [Blair98a]. However this introduces inter-ORB compatibility problems by modifying an accepted standard, OMG IDL. Moreover this approach is unsuited to the highly dynamic adaptive application areas investigated in the

Adapt project. Rather than extend IDL we instead augment stream interfaces with operations allowing their accepted media types to be queried dynamically. This alleviates the need to have access to compiled interface definitions.

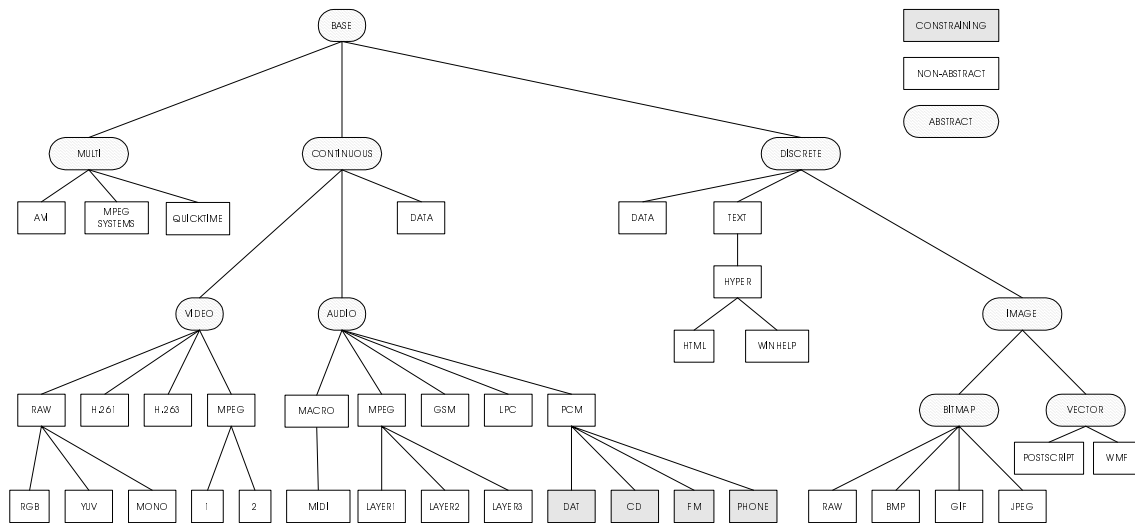


Figure 3: Example Media Type Hierarchy

The type description scheme used in our research mirrors the natural hierarchical properties of media types, as illustrated in Figure 3. A similar scheme is used in IMA MSS [IMA94a, IMA94b], which models media types as CORBA interfaces, thereby allowing a type hierarchy to be created using interface inheritance. Other media type specification schemes, such as the widely used MIME types, or the similar system used in Microsoft’s DirectShow [Microsoft98] only capture two levels of this hierarchy (e.g. “video/mpeg”).

Using IDL interfaces to model media types has drawbacks in that a CORBA invocation is required *every* time a type attribute is examined or set. This becomes unacceptable in a mobile environment characterized by low-bandwidth links. In contrast our approach allows a single invocation to retrieve a media type due to our modelling of such types as IDL structure data types. The IDL defining this is shown in Figure 4. Rather than use a fixed definition a flexible approach is taken, where a type is represented by both its type name and a list of name/value pairs holding its attributes. Compound multimedia streams (those composed of sub-streams of different types) such as MPEG Systems or Windows AVI are handled by the use of a `components` field holding the media types of all subtypes in a particular stream. The media type for MPEG Systems would, for example, use this field to hold the type of the MPEG Audio and Video streams that it contained. The use of a flexible representation method has the desirable side-effect of making the type description system highly extensible.

```

// A name/value pair type
struct pair
{
    string name;
    string value;
};

// A 'simple' media type is represented by its type name
// and a list of name/value pairs defining the attributes
struct SimpleMediatype
{
    string name;
    sequence<pair> attrs;
};

// A mediatype consists of a main type and a list of component types
// used by compound/multimedia types such as MPEG Systems and AVI
struct Mediatype
{
    SimpleMediatype type;
    sequence<SimpleMediatype> components;
};

```

*Figure 4: IDL for extensible media type definition*

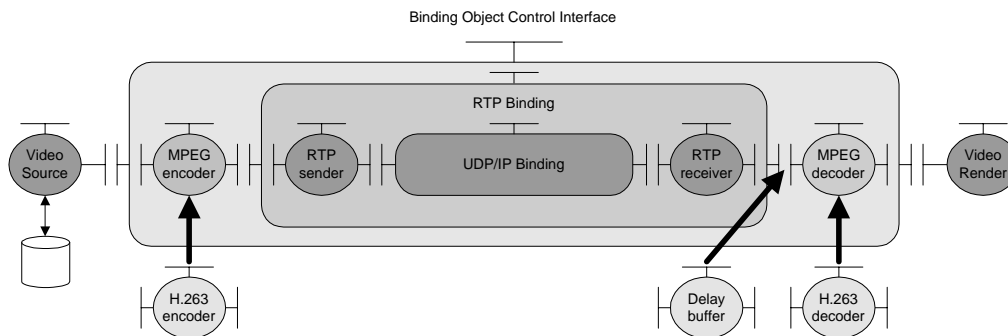
To make working with IDL Mediatype structures easier, we have developed a C++ class hierarchy which mirrors that shown in Figure 3. Methods are provided in each class to convert to/from the IDL Mediatype, thus allowing the same media type description scheme to be used throughout both CORBA and general media processing parts of a system. This avoids the need to use ad hoc type description methods in implementation.

## 5. A Simple Example

As an example of the use of object graphs to support adaptation, we consider a simple unidirectional point-to-point video binding object. This example will be referred to in forthcoming sections detailing the implementation of open bindings. This binding object is intended to encapsulate the end-to-end delivery of multimedia data and so it performs the compression, transmission and decompression of video data: at one end it consumes raw video, at the other raw video is produced for consumption by the sink object. Initially the computer is connected by a high performance fixed network and so full broadcast quality MPEG video may be sent across the network with little or no perceived degradation in quality.

At some point however the computer may move to a wireless LAN, such as WaveLAN, with greatly reduced bandwidth and increased jitter in traffic. To address this change in connectivity it is necessary to use the meta-interface of the binding object to locate the video compression object and cause it to reduce the bandwidth of compressed video. Similarly to address the increase in perceived jitter we insert a jitter compensation buffer object between the transport and video decompressor objects. At some later stage the computer moves out of range of the WaveLAN base station and

accordingly a GSM dialup link is used to reach the fixed partition of the network. Rather than change the MPEG compression parameters, it is now more prudent to replace the video compressor object with one more suited to low-bandwidth operation such as an H.263 encoder object. Similarly the decompressor object would be changed from an MPEG decompressor to a H.263 decompressor. This full range of adaptation mechanisms is illustrated in Figure 5.



*Figure 5: Example Adaptation Scenario*

More extreme adaptations would involve opening up the transport binding object to alter protocol characteristics using the aforementioned mechanisms in Ensemble. This type of adaptation would be used to achieve maximum performance over a network such as GSM where packet headers can consume a significant percentage of available bandwidth. Obviously there is a wide variation in the perceived quality of video produced by the binding object: from broadcast quality down to H.263. Nonetheless, the example shows that a constant uninterrupted video stream is possible regardless of network connectivity.

## 6. Using Open Bindings

### 6.1 The Component Class

In our current C++ implementation, every object descends from the `Component` class. This class provides the basic functionality required by extending the basic CORBA object class in several different ways:

- Firstly the `Component` class enables objects to export multiple interfaces, both stream and operational, using a mechanism similar to that used in Microsoft's COM/DCOM. This allows objects to possess multiple stream interfaces thus creating the basis for multimedia processing and interaction.

- Secondly, using this multiple interface support, the `Component` class exports a meta-interface, `MetaComponent`. This interface provides access for inspection and adaptation to the component's implementation object graph (see sections 5.2 and 5.3 below).
- Thirdly, the `Component` class provides the initial support for 'plug and play' semantics by providing methods to query the number and type of stream interfaces associated with a component; and also to ascertain whether or not a particular stream interface will accept a local binding.
- Finally, the `Component` class introduces an event mechanism allowing interested parties to register for particular events, delivered by way of a callback. One use of this is to enable monitoring components to produce events in response to QoS fluctuations. These events can then be acted upon by management components, such as the reactive objects investigated in earlier research at Lancaster [Blair98a]. Monitoring components themselves are a well-defined way of interfacing to particular QoS monitoring systems: for example the protocol layers being developed for the Ensemble protocol suite as part of the Adapt project [Coulson98], or the adaptivity-aware Mobile IPv6 protocol stack developed at Lancaster [Finney98].

## 6.2 The `MetaComponent` Class

The `MetaComponent` interface class exists to make the object graph of a component available for inspection and adaptation. This is achieved primarily by the use of a graph data structure which describes each object in the graph, and the interconnections between them. This data structure allows the programmer to traverse the graph to examine the way in which the component it represents is implemented. Using the example in section 5 above, one could ascertain that the binding object used MPEG compression objects interconnected by a transport binding object, but that no jitter-compensation component was used. This level of information allows more educated decisions on how adaptation mechanisms are to be applied. An example in this case would be the insertion of the jitter-compensation buffer component triggered by events from a network monitoring component indicating that the level of jitter on the network had increased. Section 5.3 below examines how the meta-interface of a component such as a binding object can be used to provide support for adaptation in this manner.

Apart from providing inspection and change access to a component's implementation, a `MetaComponent` interface is also responsible for policing this access to maintain consistency and security. This is aided by the indirect identification

of implementation components in an object graph: rather than use their CORBA interface references, opaque unique identifiers are used instead. The actual interface reference must be requested from the meta-interface. The particular instance of meta-interface may choose to make all components visible, or perhaps to restrict access to only certain 'safe' implementation components. Once an interface reference has been made public there is no way (in standard CORBA) to police invocations on this object, and therefore consistency may be compromised.

### **6.3 Adaptation with Open Bindings**

#### **6.3.1 Modifying Binding Behaviour**

This is the simplest form of adaptation. To modify the behaviour of a component, the application must first obtain the interface for this component via the graph traversal as described above. Once the application has obtained this interface, the programmer can make changes to that component such as increasing the delay length of the buffering component or altering the compression strategy of the MPEG component. The precise interface is clearly dependent on the object class of the component, although interface inheritance is used extensively to create a large hierarchy with many 'base' interfaces thus allowing complex components to offer increasingly specialized interfaces. An example of this is a media filtering object that, at one level, implements a basic, abstract, `MediaFilter` interface which offers a `setQualityPercentage()` method, while also defining a subclass of this interface which includes more specific operations dependent on the particular media type being filtered by that component. Polymorphism is used to allow basic controller components to manipulate other components without having to know of their specific interface type.

#### **6.3.2 Dynamically Reconfiguring Bindings**

As a component's implementation is modelled as an object graph, the obvious types of change that may be applied are the addition and removal of nodes and arcs to/from this graph. New component objects may be added to the graph; local binding arcs must then be created to connect this component to the others. Similarly it may be useful to remove whole components and their associated local bindings when they are no longer needed, an example being the jitter buffer object which would be removed when moving from wireless to fixed networks. The basic operations for manipulating object graphs are as shown in Table 1.

Operation	Effect
addObject	Adds a new object to the graph (initially unbound)
removeObject	Removes an object from the graph, breaking any associated local bindings
localBind	Create a new arc in the graph between components
breakBind	Remove an arc from the graph

*Table 1: Basic graph operations*

These graph operations provide a means of changing the functional structure of an implementation object graph. However unhindered changes could result in the functional characteristics of the binding object being destroyed. For example all component objects could be unbound leaving a disconnected graph with no path from input to output interfaces. For this reason, all changes to an implementation object graph are requested through the meta-interface of the parent component, thereby allowing it to reject those which it deems undesirable.

In some cases, however, this may prevent perfectly reasonable changes from being effected. For example, it may be desirable to replace one processing object with another. One reasonable course of action would be: (i) to add the new object to the graph, (ii) destroy the local bindings connecting the old object to other objects, and (iii) create new local bindings connecting the new processing object into the media flow allowing the graph to operate as before. However the meta-interface may reject step (ii) on the grounds that it will create a hole in the graph preventing media from flowing. There is no way for the meta-interface to know that eventually the new processing component will be connected up, therefore allowing the graph to flow again. To remedy this situation, we provide compound operations on the meta-interface (seeTable 2)

Operation	Effect
insertObject	Allows one object (offering incoming and outgoing interfaces) to be inserted between two existing objects in the graph: e.g. to insert a filtering object between CODEC and transport binding objects.
cutObject	Performs the opposite action to insert; removing a component from a pipeline and connecting predecessor and successor components directly.
replaceObject	Replaces one component with a similar component.
redirectBind	Redirects the source or sink of a local binding to an alternative stream interface, e.g. to send media data down a new path of control more suited to new network conditions.

*Table 2: Compound graph operations*

The use of compound operations allows a meta-interface to cushion or reduce the undesirable effects of these changes where appropriate. For example the replacement of a boundary component will destroy the corresponding interface mapping, therefore resulting in the local binding of an external interface being broken. The effects of this filter upwards

through the hierarchy of object graphs. If the compound `replaceObject` operation was used to replace the existing object with a new one, then this replacement could be done atomically allowing the local binding between external stream interfaces to remain intact. By defining a variety of operations, the meta-interface is able to choose the level of access that it will allow. For example a particular binding object may open up its implementation for complete inspection but only allow one particular QoS filter component to be replaced after it has ascertained that the replacement object is acceptable.

### 6.3.3 Smooth Reconfiguration

An important property of any reconfiguration process is that of *smoothness* [Mitchell98]. This can be loosely defined as the minimization of perceived disruption experienced as a result of reconfiguration. More often than not this simply means the amount of time taken for a media flow to re-establish in a stable state after changes have been made. Many applications, especially those of the video-on-demand type, need to make reconfiguration as smooth as possible.

Take the example presented in section 5 in which the encoding scheme used by a binding is changed from one type to another. This involves the removal of the existing CODEC components and the addition of their replacements. While this process is taking place no media data may flow through the binding and so, depending on the level of buffering employed by the binding, the playback of media data will be interrupted. One way to minimize this disruption time is by the use of parallel pipelines of media processing components. In this approach, rather than reconstruct an existing interconnection of components, a separate pipeline is created in to implement the new configuration. When this pipeline has been completed, the compound `redirectBind` operation is used to shift to the flow of media data from the existing configuration into the new, thus avoiding any initialization delay incurred while the new configuration is being built and connected up.

While this approach reduces delay in reconfiguration it does not eliminate it. Every pipeline of components has a certain latency associated with it: for example a pipeline that includes a jitter compensation buffer will incur at least the delay needed to fill that buffer with media data. Ongoing research in the Adapt project is addressing this problem by investigating the running of a new pipeline at the same time as the old ('hot standby'), thus allowing it to fill with media data. When media data is ready to flow out of the new pipeline, an automatic switch can be made, ensuring that reconfiguration delay is kept to an absolute minimum.

Maintaining consistency during and after such a parallel handover is an obvious concern. For example, certain video encoding schemes, such as MPEG or H.263, use different types of compressed video frame to exploit temporal redundancy: key or “I” frames (compressed independently) and “P” frames (which require a previously-transmitted “I” frame in order to be useful). If the end result of a parallel pipeline handover is that a key frame is missed then there may be a considerable delay until the video stream regains its desired state.

In addition there is the issue of time difference between pipelines: while it is trivial to ensure that media data entering both pipelines has the same temporal position, it is another matter to ensure that the streams emerging from those pipelines are exactly aligned in time so that no glitches are observed. Take for example an audio binding object in which it has been decided to add a delay buffer to protect against a rise in jitter. To achieve this in a smooth manner a separate, identical, pipeline is constructed but one which contains an additional one-second delay buffer. After this pipeline has been run in parallel, and has completely filled with audio data, the switch is made between them; the resulting observed audio flow will appear to skip back in time by one second due to the increased latency in the new pipeline.

This scenario and others like it, can often be easily overcome by careful choice of points at which to reconfigure, for example during a silence period between talk-spurts in a telephony binding. On the other extreme certain situations may require the use of time-expansion or compression methods in order to remove any temporal glitches in media flow.

## **7. Implementation Approach**

We have implemented an experimental middleware platform featuring the concept of explicit open bindings. This platform is based on a CORBA implementation from Chorus Systems, called COOL-ORB [Habert90]. This runs over a variety of computers and operating systems; our experimental testbed consists of laptop PCs running Windows NT 4.0. These are interconnected by a variety of networks, including Switched Ethernet, WaveLAN and GSM. Currently COOL-ORB supports two basic communications infrastructures: TCP/IP and CHORUS IPC. In order to support higher degree of configurability, we have extended COOL with a third communications infrastructure based on the Ensemble/Maestro protocol suite from Cornell University [Hayden96]. Further details of the extended COOL platform can be found in [Fitzpatrick98] and [Coulson98]. Continuous media communication has been optimized by efficient implementation using shared memory techniques.

To support component re-use we have implemented 'stock component' factories which allow the programmer to create a variety of commonly-used objects. These system factories are dynamically extensible, allowing new components to be made available at runtime. Building on this work we have implemented several binding object factories which construct binding objects to suit a variety of continuous media types. In particular we have investigated the application of adaptive techniques to video-on-demand and audio internet telephony applications.

## **8. Related Work**

There is currently considerable ongoing research in the area of extensible and adaptable operating systems. Key examples include Spin [Bershad95], Exokernel/Aegis [Engler90] and Spring [Mitchell94]. The aim of this work is to introduce flexibility in operating system structures to allow, for example, the addition of new services. In general, however, this research has not considered the requirements of mobile multimedia applications. In addition, we prefer to implement adaptation at a different level, i.e. in middleware. This offers a platform independent means of achieving adaptability.

Our use of object graphs is inspired by researchers at JAIST in Japan [Hokimoto96]. In their system, adaptation is handled through the use of control scripts written in TCL. Although similar to our proposals, the JAIST work does not provide access to the internal details of communication objects. Furthermore, the work is not integrated into a middleware platform. Similar approaches are advocated by the designers of the VuSystem [Lindblad96] and Mash [McCanne97]. The same criticisms however also apply to these designs. The concept of object graphs is also used in Microsoft's DirectShow architecture [Microsoft98]. This software, however, does not address distribution of object graphs. In addition, the graph is not re- configurable during the presentation of a media stream. Finally, object graphs feature in a DEIMOS, a related project at Lancaster University investigating adaptable and extensible operating systems [Clarke98].

A number of researchers have considered the impact of multimedia on middleware [Coulson95, IMA94a, IMA94b] and the impact of mobility on middleware [Davies96, Rao91]. In general, these activities do not provide as comprehensive an approach to adaptation as we feel is necessary. However researchers at CNET have developed an extended CORBA platform to support multimedia [Blair98a]; this platform features the concept of recursive bindings and has been highly influential in our research.

Finally, recent work in the OMG forum has addressed the need for multimedia streams in CORBA. In particular, a revised proposal has recently been developed in response to a Request for Proposals (RFP) for the Control and Management of Audio/Visual Streams (issued by the Telecommunication Special Interest Group). However, the current proposals do not explicitly address the issues of openness and adaptation that are the central concerns in our research.

## **9. Conclusions**

This paper has considered the design of middleware platforms to support mobile multimedia applications and has suggested that future middleware platforms should be adaptive in order to address the diverse requirements imposed by such applications. The paper has also outlined the design of an adaptive middleware platform, based on CORBA, but extended with the concepts of stream interfaces, open bindings and object graphs.

The advantages of this approach are that, firstly, applications can be made aware of arbitrary events within the platform, and, secondly, applications have a high degree of flexibility in the way they respond to events. There are also potential disadvantages with the proposed approach. For example, the programmer can be faced with added complexity in responding to events although this can be controlled by the provision of standard policies in application libraries. Secondly, there is a danger of compromising the integrity of systems by providing low level access although this problem is less serious at the middleware level than at the operating system level. We also believe that object graphs in open bindings provide a sufficiently constrained style of interaction to avoid this problem.

In addition the abstractions provided by this platform allow elegant construction of distributed multimedia systems through software component re-use. This is mainly supported by the factories described above, which allow existing implementations of common components to form part or all of a new component. Meta-interfaces are used to construct new implementation objects graphs, in the same way that they are used to adapt existing implementations. More advanced functionality, for example smooth reconfiguration strategies, can be added by specialization or subclassing. Existing binding objects can be wrapped by a new implementation which, perhaps by adding additional components, introduces new features or behaviour to the platform. In this way complex systems can be built from increasingly specialized components in an incremental manner. The widespread adoption of component-oriented multimedia programming systems such as DirectShow or the Java Media Framework highlight the need for abstractions of this type.

Finally, it could be argued that the flexibility provided by object graphs is gained at the expense of efficiency. The overheads of object graphs can however be minimized by careful engineering. This is aided by the user level implementation where most inter-object interaction is by procedure calls. In view of this, ongoing research at Lancaster will use the Adapt architecture as a basis for the investigation of complex adaptive distributed multimedia systems.

## **Acknowledgements**

The work described here is being carried out in the Adapt Project, a collaboration between Lancaster University and BT Labs sponsored by the EPSRC (Research Grant GR/K775). BT Labs are also contributing towards the cost of a PhD studentship attached to the project. Particular thanks are due to our collaborators at BT Labs namely Andrew Grace, Alan Smith and Steve Rudkin.

## References

- [Bershad95] Bershad, B.N., S. Savage, P.Przemyslaw, E.G. Sirer, M.E. Fiuczynski, D. Becker, C. Chambers, S. Eggers, S., "Extensibility, Safety and Performance in the SPIN Operating System". Proc. 15 th ACM SOSP, pp267-284, Copper Mountain CO, USA, December 1995.
- [Blair97] Blair, G.S., Coulson, G., Davies, N., Robin, P. and Fitzpatrick, T., "Adaptive Middleware for Mobile Multimedia Applications", Proc. NOSSDAV, St Louis, Missouri, USA., pp259-273, May 19-21, 1997.
- [Blair98a] Blair, G.S., J.B. Stefani, "Open Distributed Processing and Multimedia", Addison-Wesley, 1998.
- [Blair98b] Blair, G.S, Coulson, G., Robin, P., Papathomas, M, "An Architecture for Next Generation Middleare", to appear in Middleware'98, The Lake District, UK, September 15-18 1998.
- [Clarke98] Clarke, M., Coulson, G., "An Architecture for Dynamically Extensible Operating Systems". Proc. ICCDS'98, Annapolis MD, USA, May 1998.
- [Coulson95] Coulson, G., Blair, G.S., Horn, F., Hazard, L, Stefani, J.B., "Supporting the Real-Time Requirements of Continuous Media in Open Distributed Processing", Computer Networks and ISDN Systems, Vol. 27, No. 8, 1995.
- [Coulson96] Coulson, G. and Waddington, D.G., "A CORBA Compliant Real- Time Multimedia Platform for Broadband Networks", Proc. TRENDS 96, Aachen, Germany, September 1996.
- [Coulson98] Coulson, G, Blair, G.S., Davies, N, Robin, P, Fitzpatrick, T, "Supporting Mobile Multimedia Applications through Adaptive Middleware", Internal Report MPG-98-18, Distributed Multimedia Research Group, Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, June 1998.
- [Davies96] Davies, N., A. Friday, G.S. Blair, Cheverst, K, "Distributed Systems Support for Adaptive Mobile Applications", ACM Mobile Networks and Applications, Special Issue on Mobile Computing - System Services, Vol. 1, No. 4, 1996.
- [Engler90] Engler, D.R., M.F. Kaashoek, J. O'Toole jnr, J., "Exokernel: An Operating System Architecture for Application-Level Resource Management". Proc. 15th ACM SOSP pp251-266, December 1995.
- [Finney98] Finney, J, "Implementing Mobile Ipv6 for Multimedia", Proc. 1<sup>st</sup> GEMESIS Symposium on Multimedia Network Technology, Salford, UK, May 18<sup>th</sup> 1998.
- [Fitzpatrick98] Fitzpatrick, T, Blair, G.S, Coulson, G, Davies, Robin, P, "Supporting Adaptive Multimedia Applications through Open Bindings". Proc. ICCDS'98, Annapolis MD, USA, May 1998.
- [Habert90] Habert, S., L. Mosseri, V. Abrossimov, "COOL: Kernel Support for Object-Oriented Environments", Proceedings of ECOOP/OOPSLA Conference, Ottawa, Canada, October 1990.
- [Hayden96] Hayden, M., "The Ensemble System", PhD Dissertation, Dept. of Computer Science, Cornell University, USA, 1997.
- [Hokimoto96] Hokimoto, A., T. Nakajima, "An Approach for Constructing Mobile Applications using Service Proxies", Proc. 16th ICDCS'96, IEEE, May 1996.
- [IMA94a] Interactive Multimedia Association, "Multimedia System Services - Part 1: Functional Specification (2nd Draft)", IMA Recommended Practice, September 1994.
- [IMA94b] Interactive Multimedia Association, "Multimedia System Services - Part 2: Multimedia Devices and Formats (2nd Draft)", IMA Recommended Practice, September 1994.
- [Kiczales91] Kiczales, G., J. des Rivières, D.G. Bobrow, "The Art of the Metaobject Protocol ", MIT Press, 1991.
- [Ledoux97] Ledoux, T., "Implementing Proxy Objects in a Reflective ORB", Proc. ECOOP'97 Workshop on CORBA: Implementation, Use and Evaluation, Jyväskylä, Finland, June 1997.

- [Lindblad96] Lindblad, C.J. and Tennenhouse, D.L., "The VuSystem: A Programming System for Computer-Intensive Multimedia", *Journal of Selected Areas in Communications*, Vol. 14, No. 7, pp 1298-1313, IEEE, 1996.
- [Maes87] Maes, P., "Concepts and Experiments in Computational Reflection", In Proc. OOPSLA'87, Vol. 22 of ACM SIGPLAN Notices, pp147-155, ACM Press, 1987.
- [Manola96] Manola, F., "MetaObject Protocol Concepts for a 'RISC' Object Model", Technical Report TR-0244-12-93-165, GTE Laboratories, 40 Sylvan Road, Waltham, MA 02254, USA, December 1993.
- [McAffer96] McAffer, J., "Meta-Level Architecture Support for Distributed Objects", In Proceedings of Reflection 96, G. Kiczales (ed), pp39-62, San Francisco, 1996.
- [McCanne97] McCanne, S., Brewer, E., Katz, R., Rowe, L., Amir, E., Chawathe, Y., Coopersmith, A., Mayer-Patel, K., Raman, S., Schuett, A., Simpson, D., Swan, A., Tung, T-K. and Wu, D., "Towards a Common Infrastructure for Multimedia-Networking Middleware.", Proc. 7th International Conference on Network and Operating System Support for Digital Audio and Video (Nossdav'97), St Louis, Missouri, USA, 1997.
- [Microsoft98] Microsoft Corporation, "The DirectShow Software Development Kit", Microsoft Corporation 1998. Available on the Internet at <http://www.microsoft.com/directx/pavilion/dshow/default.asp>
- [Mitchell94] Mitchell, J.G., J.J. Gibbons, G. Hamilton, P.B. Kessler, Y.A. Khalidi, P. Kougiouris, P.W. Madany, M.N. Nelson, M.L. Powell and S.R. Radia, "An Overview of the Spring System". Proc. IEEE COMPCON'94, February 1994.
- [Mitchell98] Mitchell, S., Naguib, H., Coulouris, G., Kindberg, T. "Dynamically Reconfiguring Multimedia Components: A Model-Based Approach.", to appear in Proc. SIGOPS EW98, Sintra, Portugal, 7-10 September 1998.
- [Rao91] Rao, R., "Implementational Reflection in Silica", Proceedings of ECOOP'91, Lecture Notes in Computer Science, P. America (Ed), pp251-267, Springer-Verlag, 1991.
- [Schill95] Schill, A., and S. Kümmel, "Design and Implementation of a Support Platform for Distributed Mobile Computing", *Distributed Systems Engineering* Vol. 2, No. 3, pp128-141, 1995.
- [Singhai97] Singhai, A., Sane, A., Campbell, R., "Reflective ORBs: Supporting Robust, Time-critical Distribution", Proc. ECOOP'97, Jyväskylä, Finland, June 1997.
- [van Renesse96] van Renesse, R., K.P. Birman, S. Maffei, "Horus: A Flexible Group Communications Service ", *Communications of the ACM*, April 1996.
- [Watanabe88] Watanabe, T., A. Yonezawa, "Reflection in an Object-Oriented Concurrent Language", Proc. OOPSLA'88, Vol. 23 of ACM SIGPLAN Notices, pp306-315, ACM Press, 1988.
- [Yokote92] Yokote, Y., "The Apertos Reflective Operating System: The Concept and Its Implementation", Proc. OOPSLA'92, Vol. 28 of ACM SIGPLAN Notices, pp414-434, ACM Press, 1992.