

# Adaptive Middleware for Mobile Multimedia Applications

Gordon S. Blair, Geoff Coulson, Nigel Davies, Philippe Robin and Tom Fitzpatrick

Distributed Multimedia Research Group,  
Department of Computing,  
Lancaster University,  
Bailrigg,  
Lancaster,  
LA1 4YR,  
U.K.

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

## Abstract

The traditional approach to developing middleware platforms is to adopt a 'black box' philosophy whereby the platform offers a fixed programming model to applications together with a fixed per-platform implementation. In this paper we describe research which is exploring an open approach to middleware implementation. Our motivation is to accommodate the demanding requirements for quality of service adaptation which are imposed by mobile multimedia applications. We use an extended CORBA computational model which supports the concept of explicit open bindings which provide an architectural framework for openness and quality of service adaptation. The paper offers examples of the programming style facilitated by our architecture and reports on our implementation experience to date.

## 1. INTRODUCTION

Middleware has emerged as a key architectural component in supporting distributed applications. The role of middleware is to present a unified programming model to application writers and to mask out problems of heterogeneity and distribution. The importance of the topic is reflected in the increasing visibility of industrial standardisation activities such as OMG's CORBA and the Open Group's DCE.

The traditional approach to developing middleware platforms is to adopt a 'black box' philosophy whereby the platform offers a fixed programming model to applications together with a fixed per-platform implementation. A key property of this approach is that implementation details are hidden from the platform user. While this has the advantage of simplicity, there are now pressures to provide more *openness* and *adaptivity* within middleware platforms as a result of developments in the areas of multimedia and mobility. For example, multimedia applications require openness in order to extend systems to accommodate new encodings and protocols. They also require adaptivity in order to deal with changing levels of quality of service (QoS) from the underlying network or changing processing loads on machines. Mobility exacerbates these problems by

requiring applications to operate in an environment where the level of connectivity can change drastically over time (from strong connectivity, through weak connectivity, to periods of disconnection).

We believe the solution to these problems is to provide flexible middleware platforms in which applications have access to *implementations* (thus removing the black box assumption mentioned above). More precisely, applications should be able to *inspect* internal components and also *adapt* the system at run-time to meet current application needs. Such adaptation should be in terms of i) altering the behaviour of a particular component, ii) selecting a different configuration of components to provide a particular service, or iii) adding new components to the middleware platform.

*Example: Adaptation in audio-visual communication*

Consider an audio-visual application running on a mobile computer. Initially, the computer is connected by a high performance fixed network and hence it is possible to sustain high quality audio and video with QoS guarantees. Subsequently, the application observes an increase in delay jitter as the computer switches to a local area wireless network with no QoS guarantees. At this point, it is necessary to insert a buffering component to smooth out jitter. Eventually, the application observes a drastic drop in throughput as the computer is taken out of the building (requiring the use of a public wide-area radio network such as GSM). At this point, more radical steps are required, perhaps in terms of inserting an aggressive filtering component.

There has already been considerable research into specific adaptation mechanisms. For example, the Internet community have developed adaptive applications such as *vic* and *vat* to support continuous media across networks which do not support QoS. Similarly, researchers have developed a range of filtering techniques to alter the characteristics (and QoS requirements) of continuous media streams [Pasquale94, Yeadon96]. A wide range of mechanisms has also been developed in the mobile computing community; a selection of such mechanisms is

summarised in appendix A. This appendix illustrates the breadth of mechanisms available and also the fact that adaptation can take place at a variety of levels in the system. To accommodate this diversity, we believe it is important to provide an overall architectural framework offering access to the variety of mechanisms at the different levels. We believe that the middleware platform is the natural place in which to provide such a framework.

This paper describes the research carried out in the Adapt Project<sup>1</sup> which is investigating the design of adaptive middleware platforms for mobile multimedia applications. The document is structured as follows. Section 2 provides some necessary background information on CORBA. Section 3 then examines the general approach to adaptive middleware developed in the Adapt Project. In particular, this section highlights the important concept of explicit open bindings. Section 4 provides an example of use of our extended programming model. Following this, section 5 provides implementation details of a CORBA based platform based on this general approach. Some related work is highlighted in section 6 before section 7 presents some concluding remarks.

## 2. BACKGROUND ON CORBA

The Common Object Request Broker Architecture version 2.0 [CORBA96] from the Object Management Group forms the basis of our platform design. CORBA provides a technology independent object-based computational model in which distributed objects with strongly typed interfaces can be invoked with location transparency.

CORBA programmers view an object based model whereby all communication takes place across well defined *interfaces*. Objects can take either a *client* or a *server* role: clients invoke operations in the interfaces of servers. Operation invocation in CORBA is both location and implementation transparent: a client does not need to know where an object is, or the language it is written in, or the operating system or hardware which are providing its execution environment.

Interfaces to CORBA objects are described in a special purpose language called IDL (Interface Description Language). IDL, which is a superset of a subset of ANSI C++, provides constructs to define the signatures of operation interaction points plus structured data types (e.g. arrays, records, etc.) which can be used as parameters to these operations. An example IDL interface is illustrated below.

```
interface camera_ctl {
    enum state {on, off};
    enum dimension {x, y};
    struct position {x: integer, y: integer};
    readonly attribute position current_pos;
```

<sup>1</sup> The Adapt Project is a collaboration between Lancaster University and BT Labs. The work is sponsored by the EPSRC under research grant GR/K72575. A studentship associated with the project is also partly funded by BT Labs.

```
void zoom(factor: float);
void pan(dim: dimension, degrees: integer);
void switch(t: state);
}
```

To invoke a server at run time, a client must obtain a suitable *reference* to the desired server object. This reference may either be statically specified at compile time or dynamically acquired from the *Object Request Broker* (ORB) at run time. Having obtained an object reference, clients execute a *binding* step which causes the ORB to establish a communication channel between the client and the server. On completion of the binding step, the client may invoke operations on the server via a client-local *proxy* object, a pointer to which is returned by the binding step. The proxy has an identical interface to that of the server and helps realise the location transparency of the computational model. As an illustration, the following code taken from the CHORUS/COOL-ORB platform which we use in our research (see section 5) obtains a reference to a camera object (as defined above) and then invokes the `switch()` and `zoom()` operations on this object. As in a number of other CORBA platforms, the act of binding is implicit in CHORUS/COOL-ORB and is therefore not visible to the programmer.

```
CORBA_object_ptr obj;
camera_ctl_ptr c;
naming->import("obj17:cameraSrv", obj, env);
    // 'cast' obj to be used as
    // a camera_ctl proxy
c = camera_ctl::_narrow(obj);
c->switch(on);
c->zoom(1.5);
```

CORBA does provide a solution to interoperability in a heterogeneous environment. However, there are significant limitations of the design in terms of support for mobile multimedia applications [Davies96]. Firstly, there is no support for multimedia in terms of continuous media interaction or quality of service management. CORBA only supports request/ reply style operation invocation which is inappropriate for continuous media. In addition, it is not possible to specify quality of service on such interactions, e.g. in terms of bounded latencies. Secondly, as argued above, there is no support for adaptation in that the underlying implementation is completely closed. We therefore extend the CORBA architecture to support both multimedia interactions and adaptation as described below.

## 3. AN EXTENDED PROGRAMMING MODEL

### 3.1. Supporting Multimedia in CORBA

#### 3.1.1. Overview

For multimedia, a communications abstraction is required which captures the concept of information flowing over time [Blair95]. Of course, it would be possible to model continuous media flows as repeated operation invocations; with this approach however, there could be no concept of a long term information interchange governed by an overall quality of service as each invocation would be a separate, isolated event. In addition, such an implementation would be extremely inefficient as a round trip communication would be incurred for each media packet communication.

To address these requirements, we have introduced two concepts: *stream interfaces* and *explicit bindings*. These are described in turn below (further details of these extensions can be found in the literature [Blair97]).

#### 3.1.2. Stream Interfaces

Stream interfaces provide direct support for continuous media interactions. They are defined in terms of one or more *flows* where a flow is the point of production or consumption of a single continuous media type. Each flow is defined by a flow name, the type of the continuous media it handles (e.g. MPEG encoding of video) and the direction of the flow (either *out* for a producer or *in* for a consumer).

Stream interfaces are defined in terms of an extended IDL, enabling the definition of incoming or outgoing flows (denoted by the keywords *in* and *out* respectively). For example, the following extended IDL definition describes a decompressor object:

```
interface VideoDecompressor {
    typedef MPEG_T .... ; // not shown
    typedef RAW_T .... ; // not shown

    in FrameCompressed(MPEG_T frame);
    out FrameDecompressed(RAW_T frame);
};
```

Note that the types *MPEG\_T* and *RAW\_T* could be defined as any IDL type in a similar way to the arguments passed to standard IDL operations. Alternatively, to permit full generality and to accommodate types that cannot easily be expressed as IDL types (e.g. MPEG frames), we allow arguments to be object references. The intention is that an object representing a type such as *MPEG\_T* would support an operation which would return details of the encoding of the current frame as well as allowing access to the data itself.

#### 3.1.3. Objects and Interfaces

In CORBA, there is no explicit distinction drawn between objects and their interfaces as each object has exactly one interface. To permit a natural separation of concerns (e.g. a camera object may have a stream interface which emits video data and an operational interface for control purposes), we allow objects to support more than one interface along the lines of the ISO's RM-ODP model [ITU-T95]. This requires us to name both objects and interfaces using globally unique identifiers.

#### 3.1.4. Introducing Explicit Bindings

Binding is the act of establishing a logical association between objects which intend to interact. In CORBA, binding is *implicit* in that this association is established by the CORBA infrastructure and is not visible to programmers. In our platform, binding is *explicit*. In particular, bindings are *first class objects* and are created, managed and invoked in exactly the same way as other objects. Programmers are also free to develop their own binding classes, perhaps in terms of existing classes. Unlike in CORBA, bindings can be created by a third party (in CORBA, binding is always initiated by the client). We support two kinds of explicit binding: *operational bindings* and *stream bindings* (corresponding to the two styles of interface as described above). It is important to stress that bindings are *end-to-end* encompassing not just communications but also end system functions such as scheduling of incoming messages [Coulson95b]. They can also encapsulate media-specific end-system functions such as the transparent conversion of media formats, or the pre-fetching and caching of video frames.

Note that the binding model described above appears recursive in that an object must seemingly bind to a binding object using a further binding object and so on. In fact, the recursion is terminated by the provision of so-called *local bindings*. Communication across a local binding is assumed to be instantaneous and reliable; any overhead is subsumed into one or both of the objects whose interfaces are being locally bound. This normally implies that local bindings should be limited to existing within a single address space.

The binding model is illustrated in figure 1 in which two objects are shown connected by a binding object; the interfaces on the binding object are associated with the interfaces on the two objects by means of local bindings (the binding control interface is explained below).

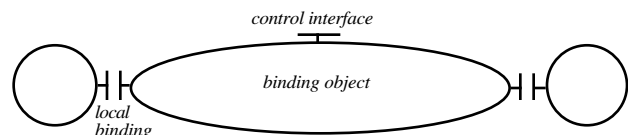


Figure 1: The binding model.

The main rationale for explicit bindings is to support *QoS management* in terms of QoS specification, monitoring and control. For stream interactions, the specified QoS governs

the ongoing flow of events (e.g. continuous media data unit arrivals/ departures) and is expressed in terms of parameters such as throughput, jitter, latency and packet loss. For operational interactions, the appropriate parameters include bounded latency and reliability (e.g. at-most-once semantics). To support QoS monitoring and control, binding objects export a *binding control interface* with operations to monitor and control the ongoing interaction. An example of such an interface for a stream binding is given below:

```
interface binding_ctl {
    enum status {ok, error};
    readonly attribute QoS currentQoS;

    status change_QoS(QoS newQoS)
    status start();
    status stop();
    // fill all buffers in binding to
    // prepare for a subsequent start()
    status prime();
    status close();
}
```

In our platform, a binding is created by invoking a new request on a binding factory object (a given platform can support a range of such factory objects). As an example, the following section of code illustrates the creation of an explicit binding between source and sink interfaces called `src_if` and `snk_if` respectively.

```
binding b = new bind_factory(reqd_type);
binding_ctl ctl = b->bind(src_if, snk_if, QoS);
status = ctl->start();
```

Note that the `bind()` operation on bindings created by a particular binding factory will support a particular schema for QoS specification. In general, the creation of a binding will only succeed if the desired quality of service can be supported by both ends of the communication and also by the underlying communications infrastructure. In addition, the participant interfaces must have compatible types. These important issues are described in more detail in [Blair97].

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.

## 3.2. The Concept of Open Bindings

### 3.2.1. Overview

In order to support adaptation, we believe it is important to allow access to the internal implementation of a binding. One way of achieving this is for the control interface on

explicit bindings to support QoS specification, monitoring and control (as described above) together with a facility for QoS re-negotiation. However, from our experience, this approach is limited in its generality as it places constraints on the possible set of actions in terms of both inspection and adaptation. This is especially problematic when mobility is introduced due to the proliferation in possible actions (for example, see Appendix A). Similarly, it is necessary to reach agreement on a standardised set of operations for QoS management, again making it more difficult to extend the functionality. Finally, this approach makes it difficult to access different levels in the implementation of the binding.

More flexibility can be provided by offering *direct access* to the implementation. It is important, however, that such access should be provided in a principled manner. We therefore introduce the concept of *open bindings* which allow the programmer to inspect and adapt the internal behaviour of a binding in a principled way. An open binding is represented as a graph of objects, connected together by *local bindings*, which together realise the required behaviour. Objects in the graph can be either processing objects or binding objects. Therefore, a graph consists of processing objects and binding objects connected together by local bindings. The use of binding objects in graphs allows open bindings to span multiple address spaces or multiple nodes.

An example of an open binding is shown in figure 2.

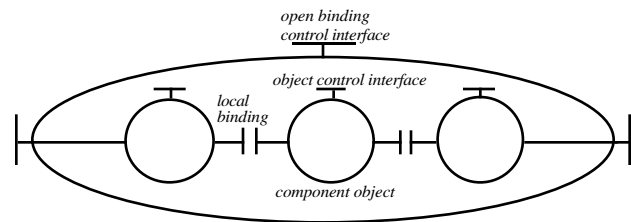


Figure 2: An open binding.

Note that binding objects in the graph 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. In addition, each object is expected to provide an interface for event notification; to use this, programmers register their interest in particular events and then receive call-backs when the events occur.

The concept of nested open binding is illustrated in figure 3 below.

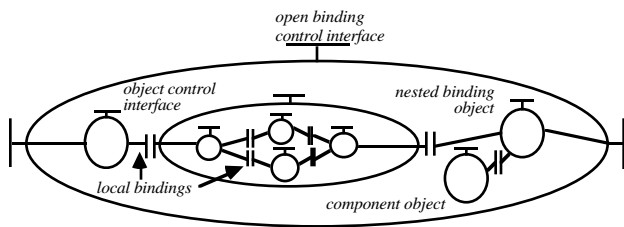


Figure 3: A nested open binding.

Standard operations are provided to manipulate this graph; our API for open bindings is described below.

### 3.2.2. An API for Open Bindings

The control interface for an open binding allows the programmer to manipulate graphs in terms of inserting new objects, replacing objects and altering configurations. Furthermore, it is possible to migrate objects to another site; this feature is particularly important in a mobile environment (cf. mobile agents [White96]).

The specification of this control interface is defined in IDL below:

```
interface open_ctl: binding_ctl {
    status {ok, error};
    typedef CORBA_object_ptr ID;
    struct LB {
        ID src_obj; ID snk_obj;
        ID src_if; ID snk_if;
    };
    typedef IDs sequence <ID, 100>;
    typedef LBs sequence <LB, 100>;

    status getConfig(out IDs objects,
                    out LBs local_bindings);
    status addObject(ID component, string name);
    status removeObject(ID obj_id);
    status getObjectByName(string name,
                           out ID obj_id);
    status localBind(ID src_if, ID snk_if,
                    out ID lb_id);
    status localUnbind(ID lb_id);
}
```

The first operation, `getConfig()`, provides access to the complete internal configuration of the open binding in terms of i) its constituent objects (i.e. processing objects and nested bindings) and ii) its local bindings. The representation of each local binding is in terms of the two interfaces involved together with the objects that 'own' those interfaces. Note that we impose the restriction that only point to point local bindings are available.

The second and third operations, `addObject()` and `removeObject()` respectively enable a component object or

nested binding to be added to or removed from a graph. `addObject()` takes the identifier of the object (plus a string name which can later be passed to `getObjectByName()`). When an object is removed by `removeObject()`, any local bindings in which the removed object was involved are implicitly broken<sup>2</sup>. The fourth operation, `getObjectByName()` does not remove the object; it simply returns a reference to the object given its string name.

The fifth and sixth operations control associations between objects inside the open binding in terms of local bindings. `localBind()` creates a local binding between two interfaces. For `localBind()` to succeed, the specified interfaces must have previously been added to the open binding using `addObject()`, must be currently unbound and must be in the same address space. `localUnbind()` simply breaks a local binding between two interfaces given a local binding identifier originally obtained from a successful call to `localBind()`.

We also assume that objects involved in open bindings support a control interface containing certain standard operations. These operations include the following:

```
struct IF {
    Type type;
    ID identifier;
};
typedef IFs sequence <IF, 100>;
status createInterface(Type type,
                      out ID identifier);
status destroyInterface(ID identifier);
status getInterfaces(IFs interfaces);
status migrate(Site site);
```

The `createInterface()` operation allows a new interface instance of a given type to be created on the object (similarly `destroyInterface()` permits an existing interface instance to be destroyed). The `getInterfaces()` operation returns a sequence of all interface instances (i.e. their types and identifiers) currently supported by the object. The `migrate()` operation requests that the object move to another site. Normally, an object will only defer to a request to migrate if it is not currently involved in any local bindings.

### 3.3.3. Discussion

Because no concurrency control is provided, there is evidently a potential danger in our API that two or more threads may simultaneously attempt to update the graph structure of an open binding and thereby destroy its integrity. However, as bindings are *per-application* entities it seems reasonable to leave it to the application to ensure that its threads do not conflict in graph modification operations. This assumption, however, may not continue to hold in a more radical open implementation framework in which

<sup>2</sup> The semantics of QoS disruption caused by breaking a local binding while data is flowing across that binding are not specified by the open binding model; rather they are determined by the objects involved in the disrupted communications path.

shared system objects such as the thread scheduler or buffer manager can be manipulated by applications. Although we intend to explore this more radical interpretation of openness in the future, we believe that opening up bindings represents a happy compromise between the conflicting requirements of application flexibility and system integrity. Open bindings are flexible in that they allow considerable control over operations on the critical data path and yet are safe in being largely isolated from one another.

Another general observation is that QoS specification in our approach can be viewed as being *procedural* rather than *declarative*. The majority of existing approaches require the programmer to declaratively state the desired behaviour, e.g. in terms of a set of desired QoS properties expressed as <name, value> pairs. Our approach, in contrast, enables the programmer to state the precise means of achieving the desired behaviour. Our argument is that the procedural approach is more flexible and scalable as more actions become available.

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 with bindings as a means of checking consistency between interfaces in the bindings. 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. EXAMPLE OF USE

To clarify the concept of open binding, we return to the example given in the introduction. For simplicity, we focus on the video binding. The initial configuration is shown in figure 4.

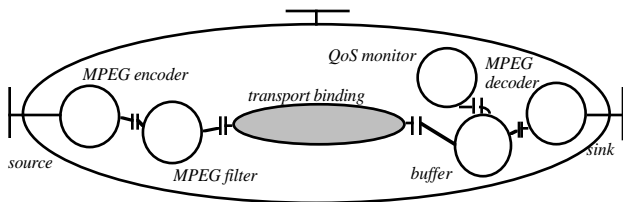


Figure 4: Initial configuration.

The application can use the QoS monitor object to observe the current levels of jitter and throughput. On detecting the initial increase in jitter (when moving to a wireless LAN), the application can then invoke the interface on the buffering object to increase the size of the internal jitter smoothing buffer.

In outline, code to do the above may appear as follows:

```
// The following registers a callback
// object with the monitor to be called
// when jitter exceeds 500ms.
monitor_id->register(callback, JITTER, 500);

// Following is executed by the handler
// of the callback object.
binding->getObjectByName("buffer", &buffer_id);
buffer_id->increase(50);
```

We assume here that the QoS monitor object supports a means of registering a callback object and associating the registration of a callback with an event threshold: i.e. the jitter has increased beyond 500ms. We further assume that the buffer object supports an operation to increase the number of buffers available for jitter correction.

To continue our example, on detecting a drastic drop in throughput (corresponding to switching to a GSM network), the application could either alter the parameters of the MPEG filter or insert a new filter object to convert MPEG to a format more suited to GSM (e.g. H.263). In addition, the transport binding could be replaced by a different object more suited to GSM, e.g. doing intelligent buffering and dialling.

Outline code to insert a new H.263 filter is as follows:

```
binding->getObjectByName("mpeg", &mpeg_id);
binding->removeObject(mpeg_id);
binding->addObject(h263filter_id, "h263filter");
h263->createInterface(MPEG_in, &h263filter_in);
h263->createInterface(H263_out, &h263filter_out);
binding->localBind(LBrec mpeg_out,
h263filter_in);
binding->localBind(LBrec h263filter_out, tp_in);
```

Here, h263 is an H.263 filter object which supports interfaces of types MPEG\_in and H263\_out. h263filter\_in and h263filter\_out, of type ID, are references to interface instances on the H.263 object. Similarly, mpeg\_out and tp\_in are references to interfaces on the MPEG encoder and transport binding object respectively.

Note finally that our approach can naturally support more advanced strategies for adaptation than those implied in this example. For example, approaches such as low latency handoff can readily be modelled in this way [Seshan95]. In addition, object migration can be used to download processing to a fixed network. This could be used, for example, to model filter propagation as used in multi-party interactions [Pasquale94].

#### 5. IMPLEMENTATION APPROACH

We are currently implementing an experimental middleware platform featuring the concept of explicit open bindings. This platform is based on a CORBA implementation from Chorus Systems, called COOL [Habert90]. This runs over a variety of computers and operating systems; our experimental testbed consists of laptop PCs running

Window-NT and Linux. These are interconnected by a variety of networks, including Switched Ethernet, Wavelan and GSM.

In order to support open bindings, the COOL platform has been extended as shown in figure 3.

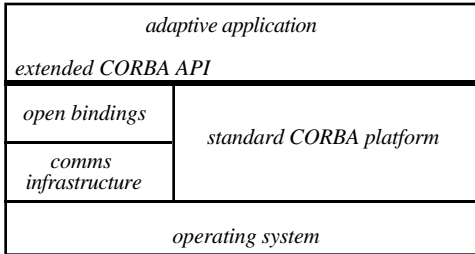


Figure 3: The extended CORBA platform.

Currently CHORUS/COOL-ORB support two communications infrastructures: TCP/IP and CHORUS IPC. In order to support higher degree of configurability, we are extending COOL with a third communications infrastructure based on Horus [van Renesse96]. Horus, developed at Cornell University, provides a modular framework for constructing protocol stacks and enables the programmer to select a particular protocol profile at bind time. Each protocol profile is built from a stack of modules each of which registers its upcalls/downcalls at connection establishment time. All inter-module communication takes place across the Horus Common Protocol Interface (HCPI) which is sufficiently rich to support a wide range of protocols. Horus offers a library of protocol modules including UDP, packet loss detection, data encryption and flow control which we are exploiting. Horus modules manipulate three types of objects: *endpoints*, *messages* and *groups*. Endpoints correspond to a communication entity which can be a thread, socket or port etc. In our terms, local bindings are also viewed as Horus endpoints. Messages are memory structures passed from one layer to another through push/pop operations. Horus groups provide sophisticated facilities for group communications which we are currently not fully exploiting but which we intend to integrate at a later stage in the research (see section 5). Crucially, all protocols in Horus, right down to the device driver level, can be implemented at the user level, thus providing the maximum potential for configurability<sup>3</sup>. We exploit this configurability by using Horus to provide a set of low level open bindings. The internal details of such open bindings are presented as object graphs, thus providing a consistent style of adaptation throughout the architecture.

The level above the communications infrastructure provides a range of (open) stream and operational bindings. This level is structured as a set of binding factories which configure Horus stacks in commonly used configurations. Through the

control interface of these bindings it is possible to reorganise Horus stacks dynamically while data is flowing on a binding. It is relatively straightforward to extend the range of bindings by constructing new binding factories from existing components. Prominent among the components are a range of filter objects for common media formats.

Currently, we have ported Horus to the Windows-NT environment and offer a small number of low level bindings through COOL. We have also constructed an example audio stream binding which demonstrates adaptation as the underlying network changed from Ethernet to Wavelan.

## 6. RELATED WORK

There is currently considerable ongoing research in the area of extensible and adaptable operating systems. Key examples include Spin [Bershad95], Exokernel/Aegis [Engler95] 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 research has been influenced by work on open implementation and reflection [Maes87, Kiczales91]. The aim of this research is to allow a program to access, reason about and alter its own interpretation. Reflection has mainly been applied in the programming language community [Kiczales91, Gowing96, Watanabe88] although some researchers have considered the application of reflection in the development of systems [Yokote92, McAffer96]. Again, however, this work has not been applied at the middleware level.

The specific concept of object graphs was introduced by researchers at JAIST in Japan [Hokimoto96]. As with our approach, a system is decomposed into a graph of objects. The system also supports a model of information flow between objects. 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. The concept of object graphs is also used in Microsoft's ActiveMovie software [Microsoft96]. 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.

A number of researchers have considered the impact of multimedia on middleware [Coulson95a, IMA94a, IMA94b] and the impact of mobility on middleware [Davies96, Schill95]. In general, these activities do not provide as comprehensive an approach to adaptation as we feel is necessary. There are some interesting developments in this area however. For example, researchers at CNET have

<sup>3</sup> Note that this effectively moves functionality from the operating system to the middleware platform thus strengthening our argument that middleware is the natural place to provide adaptation.

developed an extended CORBA platform to support multimedia [Blair97]; 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 and real-time services in CORBA. In particular, a Request for Proposals (RFP) for the Control and Management of Audio/ Visual Streams by the Telecommunication Special Interest Group [OMG,96b] and a Request for Information (RFI) from the Real-time Special Interest Group [OMG,96a] have recently closed, resulting in proposals from a number of interested parties. These activities do not explicitly address the issues of openness and adaptivity which are the central concerns in our research although our work is not incompatible with the OMG proposals.

## 7. 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 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. For example, in the scenario of section 2.3 the application could choose to either introduce a new filter or modify the behaviour of an existing filter. In many other systems, this choice would not be available; more generally, key policies are often hard-wired into the middleware platform. 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. Finally, it is arguable that the flexibility provided by object graphs is gained at the expense of efficiency. The overheads of object graphs can however be minimised by careful engineering. This is aided by the user level implementation where most inter-object interaction is by procedure calls.

Ongoing research in Adapt is looking in more detail at the support required by operational and stream bindings and also at the provision of multi-party bindings exploiting the facilities offered by Horus.

## ACKNOWLEDGEMENTS

We would like to thank our collaborators at BT Labs for their support. Particular thanks are due to Andrew Grace, Alan Smith and Steve Rudkin. Thanks also to Adrian Friday from Lancaster University for his work on the table presented as appendix A. Finally, thanks to the EPSRC for their funding of the research.

## 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". Proceedings of the 15th ACM Symposium on Operating Systems Principles, pp 267-284, Copper Mountain CO, USA, December 1995.
- [Blair97] Blair, G.S., J.B. Stefani, "Open Distributed Processing and Multimedia", To be published by Addison-Wesley, 1997.
- [CORBA96] Corba 2.0 Specification, Object Management Group Technical Document PTC/96-03-04, available at <http://www.omg.org/>
- [Coulson95a] 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.
- [Coulson95b] Coulson, G., Campbell, A., Robin, P., Blair, G., Papathomas, M. and Hutchison, D., "The Design of a QoS Controlled ATM Based Communications System in Chorus", IEEE Journal on Selected Areas in Communications, Vol. 13, No. 4 (Special Issue on ATM LANs), 686-699, May 1995.
- [Coulson96] Coulson, G. and Waddington, D.G., "A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks", Proc. International Workshop on Trends in Distributed Systems (TREDs 96), Aachen, Germany, September 1996, Springer Verlag, 1996.
- [Davies96] Davies, N., A. Friday, G.S. Blair, and K. Cheverst, "Distributed Systems Support for Adaptive Mobile Applications", ACM Mobile Networks and Applications, Special Issue on Mobile Computing - System Services, Vol. 1, No. 4, 1996.
- [Engler95] Engler, D.R., M.F. Kaashoek, J. O'Toole jnr, J., "Exokernel: An Operating System Architecture for Application-Level Resource Management". Proceedings of the 15th ACM Symposium on Operating Systems Principles, pp 251-266, December 1995.

- [Gowing96] Gowing, B., V. Cahill, "Meta-Object Protocols for C++: The Iguana Approach", In Proceedings of Reflection 96, G. Kiczales (ed), pp 137-152, San Francisco, 1996.
- [Habert90] Habert, S., L. Mosseri, V. Abrossimov, "COOL: Kernel Support for Object-Oriented Environments", Proceedings of ECOOP/ OOPSLA Conference, Ottawa, Canada, Published as SIGPLAN Notices, Vol. 25, pp 269-277, October 1990.
- [Hokimoto96] Hokimoto, A., T. Nakajima, "An Approach for Constructing Mobile Applications using Service Proxies", Proceedings of the 16th International Conference on Distributed Computing Systems (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.
- [ITU-T,95a] UIT-T, ISO/IEC Recommendation X.902, International Standard 10746-2, "ODP Reference Model: Descriptive Model", January 1995.
- [Kiczales91] Kiczales, G., J. des Rivières, D.G. Bobrow, "The Art of the Metaobject Protocol", MIT Press, 1991.
- [Maes87] Maes, P., "Concepts and Experiments in Computational Reflection", In Proceedings of OOPSLA'87, Vol. 22 of ACM SIGPLAN Notices, pp147-155, ACM Press, 1987.
- [McAffer96] McAffer, J., "Meta-Level Architecture Support for Distributed Objects", In Proceedings of Reflection 96, G. Kiczales (ed), pp 39-62, San Francisco, 1996.
- [Microsoft96] Microsoft, "Microsoft ActiveMovie: Software Development Kit", Beta Release, June 1996.
- [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, S.R. Radia, "An Overview of the Spring System". Proceedings of the IEEE 1994 Computer Conference (COMPCON '94), February 1994.
- [OMG96a] Telecom A/V RFP:  
[http://www.omg.org/library/schedule/Telecom\\_RFP1.htm](http://www.omg.org/library/schedule/Telecom_RFP1.htm)
- [OMG96b] Realtime RFI:  
[http://www.omg.org/library/schedule/Realtime\\_RFI.htm](http://www.omg.org/library/schedule/Realtime_RFI.htm)
- (<ftp://ftp.omg.org/pub/docs/orbos/>)
- [Pasquale94] Pasquale, J., Polyzos, G., Anderson, E., Kompella, V., "Filter Propagation in Dissemination Trees: Trading Off Bandwidth and Processing in Continuous Media Networks", Proceedings of the 4th International Conference on Network Support for Digital Audio and Video, Lancaster, November 1993, Published in Shepherd, D., Blair, G.S., Coulson, G., Davies, N., Garcia, F. (Eds), Lecture Notes in Computer Science, Vol. 846, Springer Verlag, 1994.
- [Schill95] Schill, A., and S. Kümmel, "Design and Implementation of a Support Platform for Distributed Mobile Computing", Distributed Systems Engineering Vol. 2, No. 3, pp 128-141, 1995.
- [Seshan95] Seshan, S., "Low Latency Handoff in Wireless Networks", PhD Thesis, University of California, Berkeley, 1995.
- [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", In Proceedings of OOPSLA'88, Vol. 23 of ACM SIGPLAN Notices, pp 306-315, ACM Press, 1988; Also available as Chapter 3 of "Object-Oriented Concurrent Programming", A. Yonezawa, M. Tokoro (eds), pp 45-70, MIT Press, 1987.
- [White96] White, J.E. "Mobile Agents", In Software Agents, J. Bradshaw (Ed), MIT Press, 1996.
- [Yeadon96] Yeadon, N., Garcia, F., Shepherd, D., Hutchison, D., "Filters: QoS Support Mechanisms for Multipeer Communications, To appear in IEEE Journal on Selected Areas in Communications, Special Issue on Distributed Multimedia Systems and Technology, 1996.
- [Yokote92] Yokote, Y., "The Apertos Reflective Operating System: The Concept and Its Implementation", In Proceedings of OOPSLA'92, vol. 28 of ACM SIGPLAN Notices, pp 414-434, ACM Press, 1992.

## APPENDIX A: A SELECTED RANGE OF ADAPTATION MECHANISMS

Level	Technique	Description
User	Change of working practices	The user can alleviate demands on the network, e.g. change task, swap from synchronous to asynchronous collaboration or specify which tasks are most important to them.
Application	Restructure using agents or delegation of processing	Processing/network intensive tasks can be offloaded to remote sites or pre-processing or filtering applied to remote data (reducing bandwidth requirements and freeing host for other tasks/dozing to save power).
	Use proxy services	The application can use local substitute services based on cached information (often with reduced functionality) while disconnected.
	Change model of interaction	Interactions can be adjusted from polling to event based structures or from RPC to an alternative (perhaps asynchronous) paradigm.
	Reorganise application structure	One example of application restructuring is to change from using distributed state to a centralised architecture to simplify consistency management in unreliable conditions.
	Re-bind to new services	The application may be able to rebind to equivalent services which are easier/cheaper to access. Alternatively, it may be possible to migrate the service or application component.
	Change application demands	New QoS requirements can be negotiated or non-essential bindings dropped. Alternatives may be possible, e.g. lossy encoding.
	Adjust consistency requirements	Groups may be able to tolerate weaker consistency or adjust operations to achieve quorum, yet avoid hard to reach members.
Distributed system	On-demand cache management	Information can be fetched only when needed, instead of speculatively, e.g. opening the first page of a document and transferring successive pages later, or retrieving e-mail headers before message bodies.
	Prefetching into the cache	The application can fetch information while the link is good, in case it is required when the link degrades or becomes expensive.
	Apply filtering and compression	The volume of information to transfer can be reduced by compression or filtering non-essential frames from hierarchically encoded data.
	Efficient protocol utilisation of the channel	The transport mechanisms can be adjusted to match channel characteristics, e.g. retransmission/backoff strategies, header compression, error control and handling of asymmetric channels.
Transport and below	Change or introduce new protocols	New protocols can be selected which suit the characteristics of a particular network or appropriate protocols can be introduced (e.g. injecting a reliable data link layer).
	Optimise data for the network	Protocols can adjust their packet sizes to suit different networks. The operating system can adjust the queue sizes onto the network interfaces which impacts on latency, particularly of multimedia streams.
	Optimisation of multicast	Multicasts can be mapped onto the network technology, particularly those with partial or full hardware multicast support.
	Optimise for the characteristics of the network	There are a number of cost and network structure optimisations. For instance, batching data to spread the dialling delays, or transferring additional information while the time is already paid for.
	Reordering of data	The priority or urgency of data may require that it is handled preferentially in scarce bandwidth situations.
	Demultiplexing to multiple networks	If multiple technologies are available simultaneously, it may be advantageous to use several at once.