

Supporting Mobile Multimedia Applications through Adaptive Middleware

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

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

E-mail: [geoff, gordon, 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 fixed per-platform implementations. In this paper we describe research which is exploring an open approach to the implementation of middleware platforms. Our motivation is to accommodate the demanding requirements for quality of service adaptation as imposed by mobile multimedia applications. We use an extended CORBA computational model featuring the concept of explicit open bindings. This provides an architectural framework for openness and quality of service adaptation. The paper concludes by considering the more general application of an open systems philosophy; in particular, we introduce the concept of reflective middleware.

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 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 Microsoft's DCOM.

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 fixed per-platform implementations. 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 controlled 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 particular components, ii) selecting a different configuration of components to provide a particular service, or iii) adding new components to the middleware platform.

Note that 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 [1, 2]. A wide range of mechanisms has also been developed in the mobile computing community. A selection of such mechanisms is summarised in the appendix. 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 these diverse strategies, we believe it is important to provide an 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 research carried out in the Adapt Project¹ which is investigating the design of adaptive middleware platforms for mobile multimedia applications. The document is structured as follows. Section 2 describes the general approach to adaptive middleware developed in the Adapt Project, with particular emphasis on *open bindings*. Section 3 then provides further detail on the use of open bindings for inspection and adaptation. Following this, section 4 provides implementation details of a CORBA based platform based on this general approach. Section 5 then extrapolates from the results of Adapt and proposes some interesting avenues for future research. Related work is highlighted in section 6 before section 7 presents some concluding remarks.

2. The overall architecture

2.1. Motivation

Our design is based on the Common Object Request Broker Architecture version 2.0 [3] from the Object Management Group. While CORBA provides a solution to interoperability in a heterogeneous environment, it has significant limitations in terms of support for mobile multimedia applications [4]. 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.

2.2. Introducing explicit bindings

In order to address our requirements, we extend the programming model offered by CORBA. In particular, we introduce the concept of *explicit binding*, as proposed in for example [5, 6, 7]. In the current CORBA programming model, binding is implicit in that, when objects interact, an appropriate communications path is

created transparently by the underlying ORB. In our approach, we suggest that bindings should be created explicitly by the programmer; the result is then the creation of 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). Another consequence is that group communication is naturally supported, i.e. bindings can be either point-to-point or multiparty (as will be seen later, this is an important aspect of our design).

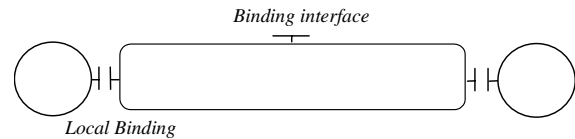


Figure 1: Explicit binding.

More generally, the importance of explicit binding is twofold:

- i) the act of creating a binding can subsume *static* QoS management functions such as negotiation, admission control and resource reservation, and
- ii) 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.

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. For operational bindings, 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. 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,

¹ 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.

programmers are free to develop their own binding classes, perhaps in terms of existing classes.

Note that, in general, the changes described in this section bring CORBA into closer alignment with the ISO Reference Model for Open Distributed Processing (RM-ODP). Indeed, our work is strongly motivated by results from this community. For further discussions of the proposed extensions and also the work on RM-ODP please refer to [8].

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.

2.3. From explicit binding to open binding

General approach. To adequately support multimedia computing, it is necessary for the application to be able to exert some control over bindings. One way of achieving this is for the 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 (see the appendix). Our approach is for bindings to offer an interface providing access to the implementation of the binding in terms of an *object graph*, representing the underlying end-to-end communications path. This can be viewed as a procedural as opposed to a declarative approach to QoS management [9]. 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 [10]. 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.

Object graphs. An object graph consists of processing objects and (nested) binding objects connected together

by *local bindings*². 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. Examples of management functions are violation detectors, statistics gatherers and reconfiguration policies.

To control the visibility of interfaces within a binding, we introduce the concept of *interface mapping*. Interface mapping allows an external interface to map directly 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, nested 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. 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 (see section 3.2).

The concept of (nested) open bindings is illustrated in figure 2.

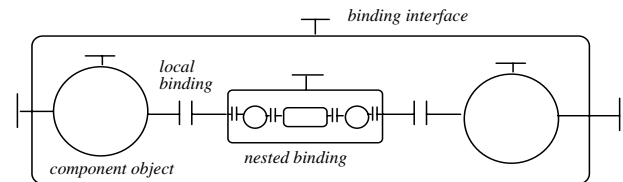


Figure 2: Nested open bindings.

² Local bindings are required to break the potential recursion in the binding model; without local bindings, it would be necessary to bind a binding to an object using a binding, and so on.

3. Using open bindings

3.1. The component class

In our C++ based environment, open bindings are created as a subclass of a `Component` object class (similar frameworks can be provided for other programming languages). This class enhances CORBA objects in three important ways. Firstly, the `Component` class enables CORBA objects to be created with multiple interfaces (stream or operational interfaces) as required by the extended programming model (see section 2.2 above). Secondly, the class provides access to an interface which supports operations to inspect and adapt the associated object graph structure (see sections 3.2 and 3.3 below). Thirdly, the `Component` class allows the programmer to query the various interfaces supported by the component, and to ascertain whether a given interface will accept a local binding, e.g. whether it will accept a given media type.

3.2. Support for inspection

Object graphs offer a powerful and intuitive way for the programmer to inspect the implementation of open bindings. The `Component` interface referred to above provides one operation, `getConfiguration`, which returns a representation of the object graph, which consists of a list of components and a list of local bindings. By traversing this graph structure, the programmer can identify how the individual components are connected together. This level of information enables more informed decisions on how adaptation mechanisms are to be applied, e.g. the removal of a buffering component to reduce end-to-end latency on detecting a drop in the level of jitter.

Every component in the graph also has a unique name which can be used to access the object directly, using a `getObjectByName` method.

Once accessed, a component in the graph can be queried with regard to its type, its location and the number of stream interfaces that it possesses. As mentioned above, components also support an event notification mechanism. An application can register for events from a particular component and then receive a call-back when appropriate events occur. This feature can be used, for example, to allow an application to monitor the level of packet loss experienced over the current network.

3.3. Support for adaptation

As mentioned above, two approaches to adaptation are supported. Firstly, an application can modify the behaviour of a component (without altering the structure of the object graph). To do this, the application must first obtain the relevant interface (as discussed above) and then make appropriate changes such as increasing the size of a buffering component or altering the compression strategy of an MPEG component.

Secondly, an application can (dynamically) adapt the behaviour of an open binding through direct manipulation of the object graph. For example, it is possible to add new components and/ or new local bindings to the graph. Similarly it may be useful to remove components and their associated local bindings when they are no longer needed. For example, a jitter-compensation buffer could be removed when moving from a wireless to a fixed network.

Dynamic reconfiguration raises a number of important issues, particularly with respect to maintaining *consistency* of the associated binding. This problem is partly addressed by the strong type checking incorporated in our approach (and in RM-ODP in general). Furthermore, type information is available at run-time through the inspection capabilities as discussed above. A more sophisticated solution is required however when multiple changes are required to return the graph to a consistent state, e.g. when changing both a compression and a decompression component. Furthermore, it should be possible to make such changes and preserve the *smoothness* of the ongoing interaction [11]. Further details of our approach to consistency management can be found in [12].

4. Implementation details

4.1. Overview

We have implemented an experimental middleware platform featuring the concept of explicit open bindings. This platform is based on a CORBA implementation from Sun Microsystems, called COOL-ORB [13]. COOL-ORB runs over a variety of computers and operating systems; our experimental testbed consists of laptop PCs running Window-NT. 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.

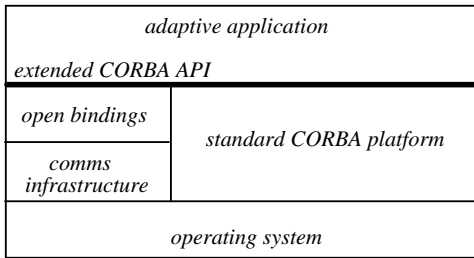


Figure 3: The extended CORBA platform.

The communications infrastructure is implemented using an extended version of the Ensemble communications framework as discussed in detail below. The open bindings layer then exploits the flexibility provided by the resultant communications infrastructure by offering a set of open and extensible binding factories. More specifically, we provide a set of low level (operational and stream) binding factories mapping directly on to the facilities offered by Ensemble. The key role of such factories is to maintain the uniform abstraction of object graphs, masking out the different mechanisms for adaptation provided by different communication stacks. Note that particular constraints are imposed on object graphs implemented using Ensemble; for example, object graphs are restricted to linear stacks. This is not ideal. For example, it is often useful to represent communication stacks as more general graphs. In addition, this approach mixes service and management in the same linear stack. We are planning to remove this restriction in our future research in this area (as described in section 5).

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 [2].

4.2. Basic communication infrastructure

Background on Ensemble. Currently CHORUS/COOL-ORB support two communications infrastructures: TCP/IP and CHORUS IPC. In order to support a higher degree of configurability, we have extended COOL with a third communications infrastructure based on Ensemble/ Maestro [14]. Ensemble, developed at Cornell University, is the successor to Horus [15]. The software provides a modular framework for constructing protocol stacks, specifically protocol stacks for group communication (point-to-point communication is treated as a special case). The role of Maestro is to offer a C++ interface to Ensemble.

Protocol stacks are constructed using a suite of *micro-protocols* (examples of micro-protocols include group membership, fragmentation and re-assembly, flow control and rate control). Ensemble enables the programmer to select a particular *protocol profile* at bind time by providing a list of the component micro-protocols. For example, the following protocol profile provides a stack offering virtual synchronous group communication:

```
vsync = [Gmp; Sync; Heal; Migrate; Switch;
Adaptor; Frag; Suspect; Flow]
```

Micro-protocols are implemented in a dialect of ML, Objective Caml [16], which allows them to be written in a very compact and declarative manner. Micro-protocols in a given stack communicate using *events* as a generalisation of control and data passing in traditional communications stacks. Each micro-protocol is implemented as a set of handlers which manipulate events going up or down the stack. The handlers then operate by consuming events, processing events and generating new events (some handlers simply pass on the event to the next micro-protocol). A given micro-protocol can also define headers to add to messages (headers generated by one layer are not interpreted by other layers).

Integration with COOL. Integration between COOL and Ensemble is achieved as follows. COOL provides a framework based on *comChannels*, which represent alternative protocol stacks. A generic *comManager* selects an appropriate *comChannel* based on an object's *comId*. Given this framework, we provide new *comChannel* classes, which inherit from appropriate Maestro classes. Using Ensemble, *comChannels* are implemented as a group of Ensemble *endpoints*, allowing multicasting of requests.

Building on this, we have modified the COOL Basic Object Adapter so that server implementations can specify which communications protocols should be used (by default, this is TCP/IP). In particular, we have modified the `COOL_bind` operation on the Basic Object Adapter as follows:

```
void COOL_bind(
    const ObjImpl& server,
    Obj_ptr& inter,
    COOL_ComCtrl_ptr& ctrl,
    const char *comName,
    CORBA_Environment& _env);
```

The role of this operation is to enable a client to bind to a particular server before invoking (possibly remote) operations. The first two parameters respectively specify the object implementation and pointer to be used to invoke this object (as in the standard `COOL_bind`). The

third parameter then represents the control interface for the newly instantiated protocol stack. The `comName` parameter specifies the preferred protocol suite to be used, followed by a specification of a protocol stack (if Ensemble). The final parameter is a standard CORBA environment variable.

As an example, the following call registers `serverImpl` with the Basic Object Adapter and associates with it an appropriate Ensemble stack:

```
COOL_bind(serverImpl, server,
ctrlIntf,
"Ensemble::Gmp:Sync:Heal:Switch:Frag
:Suspect:Flow", env);
```

Importantly, the control interface is used to perform adaptation on the resultant stack as discussed below.

4.3. Key extensions

Supporting Adaptation in Ensemble. Ensemble currently supports adaptation only at a coarse granularity, i.e. it is possible to switch to a new protocol stack at runtime. In our framework, this is achieved by invoking a `changeProperties` operation on the control interface mentioned above. (This operation is used to implement the direct manipulation style of adaptation on object graphs, with the constraint that the programmer must replace one entire graph structure with another.)

Within Ensemble, such reconfigurations are generally managed by a *group coordinator* and handled by a *Protocol Switch Protocol* (PSP). The PSP manages synchronization issues among group participants. This operation is not trivial as some members can be temporarily disconnected or crashed. A detailed explanation of the PSP protocol can be found in [17].

Furthermore, to support the component modification style of adaptation, we have added a `changeParameters` operation to the control interface whereby new parameters can be specified for a given micro-protocol layer. This operation takes a string as parameter representing the new values for different micro-protocols. For example, the following call changes the value of the `interval` parameter of the `rate` micro-protocol to 1.0 (time is the type of the parameter):

```
changeParameters (
    "rate_interval = 1.0:time");
```

Changes are then translated into a set of events which are passed down the stack and interpreted/ absorbed by appropriate layers. To accommodate this new functionality it is also, of course, necessary to make changes to the various micro-protocol layers to interpret such events.

Secondly, we have developed an *adaptor micro-protocol* to monitor and adapt group communication stacks. The first task of this micro-protocol is to collect statistics on the performance of each member of the group. The second task is then to detect degradations that may necessitate actions like changing parameters of a given micro-protocol or reconfiguring the protocol stack. This is done by periodically executing a number of *adaptation policies* (see below). The overall aim of these policies is to maintain a specific level of QoS. In the most general case, a given stack can have multiple adaptor micro-protocols at different levels in the stack together with multiple policies within each adaptor instance. Should an adaptor fail to deal with a specific degradation, it will be passed on to other micro-protocols and possibly eventually to the application.

Adaptation policies. Policies are written as ML functions, and take a *state* structure and a *view* as parameters. The state structure contains a performance profile of each group member. This includes throughput obtained, the number of messages lost, disconnection time, etc. The view parameter contains the list of members currently in the group. A policy then returns a list of (up or down) events that will be passed on to other layers. The signature of such function is as follows:

```
val policy : 'state -> 'view
    -> Event.t list
```

As an example, consider the case where throughput drops significantly on changing over to a slow network. In such a situation, various actions are possible:

- decrease the frequency at which the Ensemble *suspect layer* pings this member before marking this member as disconnected,
- switch to sending messages in unreliable mode,
- compress headers and/or messages, or
- insert a filtering layer in the protocol stack.

Each of these would be initiated in the same way, simply by generating the appropriate event.

Policies can either be statically linked with Ensemble or written separately and dynamically loaded as bytecode object files. The latter option gives more freedom to application writers as they can dynamically extend the set of policies without having to recompile/link the application.

5. Future work

Our experiences in the Adapt Project have indicated that open bindings provide the necessary level of inspection and adaptation to meet many of the needs of mobile and multimedia applications. However, the Adapt

architecture is restricted to openness with respect to communications (i.e. bindings). We argue that similar levels of openness are required in other areas of the system, e.g. thread scheduling, buffer management, marshalling, and dispatching. Consequently, we are now pursuing the more general application of inspection and adaptation to the complete middleware architecture. In particular, we are investigating the role of *open implementation* and *reflection* in the design of middleware platforms.

The concept of open implementation has been investigated by a number of researchers, most notably Kiczales et al at Xerox PARC [18]. 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. Importantly, they argue that there should be a *principled* division between the *base interface* of a module and its *meta-interface*. This approach is neatly captured by Rao [19]:

"A system with an open implementation provides (at least) two linked interfaces to its clients, a base-level interface to the system's functionality similar to the interface of other such systems, and a meta-level interface that reveals aspects of how the base-level interfaces is implemented".

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 [20], 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, secondly, modification of this self representation can result in modification or extension of the execution environment. In other words, a reflective system naturally supports inspection and adaptation.

In our opinion, this approach provides a strong basis for the design of future middleware platforms and overcomes the inherent limitations of technologies such as CORBA. In particular, reflection provides the opportunity to have principled and comprehensive access to the engineering of a middleware platform. This compares favourably with CORBA which, as stated above, generally follows a black box philosophy with minimal, ad hoc access to internal details.

The reflective approach also generalises the viewpoints approach to structuring advocated by RM-ODP [8]. RM-ODP distinguishes between the Computational Viewpoint (focusing on application-level

objects and their interaction) and the Engineering Viewpoint (which considers their implementation in a distributed environment). This approach generally enforces a two-level structure, in that it is not possible to analyse engineering objects in terms of their internal structure or behaviour.

Some early results from our research on reflective middleware can be found in [21, 22].

6. Related Work

Our use of object graphs is inspired by researchers at JAIST in Japan [23]. 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 [24] and Mash [25]. The same criticisms however also apply to these designs. Microsoft's DirectShow software also uses object graphs, but does not address the distribution of object graphs. In addition, the graph is not re-configurable at run-time.

Focusing on Ensemble, van Renesse et al [17] describe how to perform adaptation using *violation detectors* (implemented as micro-protocols). Such detectors can be placed at different levels in the protocol stack. When a violation is detected, a protocol switch request is initiated. Hiltunen and Schlichting [26] propose a similar approach in the context of the x-kernel. Our approach extends this work by including QoS policies which can not only switch protocol stacks but also reconfigure micro-protocols in an existing stack (in all cases, based on collected statistics).

More generally, a number of researchers have considered the impact of multimedia on middleware [27, 28, 29] and the impact of mobility on middleware [30, 4]. However, these activities do not provide as comprehensive an approach to adaptation as we feel is necessary. Nevertheless, there are some interesting developments in this area. For example, researchers at CNET have developed an extended CORBA platform to support multimedia [8]; this platform also features the concept of recursive bindings and has been influential in our research. In addition, 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 address the issues of openness and adaptation that are the central concerns in our research.

There is also growing interest in the use of reflection in distributed systems. Pioneering work in this area was carried out by McAffer [31]. With respect to middleware, researchers at Illinois have carried out initial experiments on reflection in Object Request Brokers (ORBs) [32]. 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 for groups or, more generally, bindings. Researchers at APM have developed an experimental middleware platform called FlexiNet [33]. This platform allows the programmer to tailor the underlying communications infrastructure by inserting/ removing layers (using a meta-interface). Their solution is, however, language-specific, i.e. applications must be written in Java. Finally, researchers at the Ecole des Mines de Nantes are also investigating the use of reflection in proxy mechanisms for ORBs [34].

7. Concluding remarks

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 described the design of an adaptive middleware platform, based on CORBA, but extended with the concepts of open bindings and object graphs (providing a uniform approach to inspection and adaptation at all layers). The resultant implementation exploits the functionality of Ensemble to provide fine-grained (re-) configurability of the underlying protocol stack. Importantly, we extend Ensemble with adaptor micro-protocols that support the collection of performance statistics and initiate re-configurations based on adaptation policies.

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 group facilities offered by Ensemble. More fundamentally, we are also continuing research on reflective middleware platforms, applying open implementation principles to all aspects of a middleware architecture.

Acknowledgements

The research described in this paper is funded by the EPSRC together with BT Labs (Research Grant GR/K72575). The authors would also like to acknowledge the contributions a number of researchers at Lancaster to the ideas described in this paper, namely Anders Andersen (also of the University of Tromsø,

Norway), Lynne Blair, Mike Clarke, Fabio Costa, Hector Duran, Lee Johnston, Michael Papatomas, Nikos Parlavantzas, and Katia Saikoski.

References

- [1] Pasquale, J., Polyzos, G., Anderson, E., and 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.
- [2] Yeadon, N., Garcia, F., Shepherd, D., and Hutchison, D., "Filters: QoS Support Mechanisms for Multipeer Communications, IEEE Journal on Selected Areas in Communications, Special Issue on Distributed Multimedia Systems and Technology, Vol. 14, No. 7, pp 1245-1262, September 1996.
- [3] Corba 2.0 Specification, Object Management Group Technical Document PTC/96-03-04, available at <http://www.omg.org/>, 1996.
- [4] Davies, N., A. Friday, G.S. Blair, and 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.
- [5] ITU-T/ ISO/IEC Recommendation X.902, International Standard 10746-2, "ODP Reference Model: Descriptive Model", January 1995.
- [6] Architecture Projects Management Ltd., "ANSAware 4.1, Application Programming in ANSAware", February 1993.
- [7] Bond, A., Arnold, D., Chilvers, M., "Designing and Building an ODP Environment", Proceedings of the IFIP International Conference on Open Distributed Processing and Distributed Platforms (ICODP/ICDP), Toronto, Canada, May 1997.
- [8] Blair, G.S., and Stefani, J.B., "Open Distributed Processing and Multimedia", Addison-Wesley, 1997.
- [9] Blair, G.S., Coulson, G., Davies, N., Robin, P. and Fitzpatrick, T., "Adaptive Middleware for Mobile Multimedia Applications", Proc. NOSSDAV, St Louis, Missouri, USA., pp 259-273, May 19-21, 1997.

- [10] 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.
- [11] Mitchell, S., Naguib, H., Coulouris G., and Kindberg, T., "A QoS Support Framework for Dynamically Reconfigurable Multimedia Applications", Proc. DAIS 99, the Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems, Helsinki, Finland, June 1999.
- [12] Fitzpatrick, T., Blair, G.S., Coulson, G., Davies, N., and Robin, P., "A Software Architecture for Adaptive Distributed Multimedia Systems", IEE Proceedings on Software, Vol. 145, No. 5, pp 163-171, October 1998.
- [13] Habert, S., L. Mosseri, and Abrossimov, V., "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.
- [14] Hayden, M., "The Ensemble System", PhD Dissertation, Dept. of Computer Science, Cornell University, USA, 1997.
- [15] van Renesse, R., K.P. Birman, and Maffeis, S., "Horus: A Flexible Group Communications Service", Communications of the ACM, April 1996.
- [16] Leroy, X., and Weis, P., "Manuel de Référence du Langage Caml", Inter Editions, Paris, 1993.
- [17] van Renesse, R., Birman, K., Hayden, M., Vaysburd, A., and Karr, D., "Building Adaptive Systems using Ensemble", Technical Report TR 97-1638, Cornell University, USA, July 1997.
- [18] Kiczales, G., J. des Rivières, and Bobrow, D.G., "The Art of the Metaobject Protocol", MIT Press, 1991.
- [19] Rao, R., "Implementational Reflection in Silica", Proceedings of ECOOP'91, Lecture Notes in Computer Science, P. America (Ed), pp 251-267, Springer-Verlag, 1991.
- [20] Maes, P., "Concepts and Experiments in Computational Reflection", In Proceedings of OOPSLA'87, Vol. 22 of ACM SIGPLAN Notices, pp147-155, ACM Press, 1987.
- [21] Costa, F.M., Blair, G.S., and Coulson, G., "Experiments with Reflective Middleware", Proceedings of the ECOOP'98 Workshop on Reflective Object-Oriented Programming and Systems, Brussels, July 1998.
- [22] Blair, G.S., Coulson, G., Robin, P., and Papatomas, M., "An Architecture for Next Generation Middleware", Proc. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), Kluwer, September 1998.
- [23] Hokimoto, A. and Nakajima, T., "An Approach for Constructing Mobile Applications using Service Proxies", Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS'96), IEEE, 1996.
- [24] 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.
- [25] 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.
- [26] Hiltunen, M.A., and Schlichting, R.D., "Adaptive Distributed and Fault-Tolerant Systems", International Journal of Computer Systems Science and Engineering, Vol. 11, No. 5, pp 125-133, September 1996.
- [27] Coulson, G., Blair, G.S., Horn, F., Hazard, L, and 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.
- [28] Interactive Multimedia Association, "Multimedia System Services - Part 1: Functional Specification (2nd Draft)", IMA Recommended Practice, September 1994.
- [29] Interactive Multimedia Association, "Multimedia System Services - Part 2: Multimedia Devices and Formats (2nd Draft)", IMA Recommended Practice, September 1994.

- [30] Schill, A., and Kümmel, S. "Design and Implementation of a Support Platform for Distributed Mobile Computing", *Distributed Systems Engineering* Vol. 2, No. 3, pp 128-141, 1995.
- [31] McAffer, J., "Meta-Level Architecture Support for Distributed Objects", *Proceedings of Reflection'96*, G. Kiczales (ed), pp 39-62, San Francisco; Also available from the Department of Information Science, University of Tokyo, 1996.
- [32] Singhai, A., Sane, A. and Campbell, R., "Reflective ORBs: Supporting Robust, Time-critical Distribution", *Proc. ECOOP'97 Workshop on Reflective Real-Time Object-Oriented Programming and Systems*, Jyväskylä, Finland, 1997.
- [33] Hayton, R., "FlexiNet Open ORB Framework", *APM Technical Report 2047.01.00*, APM Ltd, Poseidon House, Castle Park, Cambridge, UK, 1997.
- [34] Ledoux, T., "Implementing Proxy Objects in a Reflective ORB", *Proc. ECOOP'97 Workshop on CORBA: Implementation, Use and Evaluation*, Jyväskylä, Finland, 1997.

Appendix: A Selection 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.

