

Design and Application of TOAST: an Adaptive Distributed Multimedia Middleware Platform

Tom Fitzpatrick¹, Julian Gallop², Gordon Blair¹, Christopher Cooper², Geoff Coulson¹, David Duce³ and Ian Johnson²

¹Computing Department, Lancaster University, Lancaster, LA1 4YR UK.

²CLRC Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire, OX11 0QX UK

³Oxford Brookes University, Gipsy Lane, Oxford, OX3 0BP UK

¹{t.f.|gordon|geoff}@comp.lancs.ac.uk

²{J.R.Gallop|C.S.Cooper|I.J.Johnson}@rl.ac.uk

³daduce@brookes.ac.uk

Abstract. The rise of mobile computing and wireless network technology means that, increasingly, applications must adapt to their environment, in particular network connectivity and resource availability. This paper outlines the TOAST middleware platform which provides component-oriented CORBA support for adaptive distributed multimedia applications. In particular, the paper examines how the areas of reflection and open implementation have shaped the approach to adaptation support in TOAST. The paper then discusses novel ongoing research which is investigating middleware support for distributed cooperative visualization using TOAST as a base.

1. Introduction

The modern computer user is increasingly likely to be mobile. With the advent of powerful laptop computers, personal digital assistants (PDAs) and, more importantly, wireless networking technology, mobile computing is a fast-growing area. Understandably, the mobile computing user expects applications to function regardless of mobility issues, in the same way that cellular voice or fax services are (generally) expected to be available regardless of location. To achieve this ideal, there are, unsurprisingly, many challenges to be overcome.

Perhaps the greatest hurdle for mobile computing is that the Quality of Service (QoS) differences between mobile network types make transparent use of most networked applications impossible. Today, a mobile user may roam from a desktop scenario with a 10 or (more commonly) 100Mbit/s Ethernet connection, round a local site with 2-10Mbit/s Wireless LAN connectivity or in the wide-area with low-bandwidth GSM dialup or TETRA wireless link. In addition to connectivity changes, resource availability may also vary. For example, a laptop computer may reduce its processor power to increase battery life when mobile. In addition, PC-CARD devices such as media capture or compression cards may need to be removed to make room for network interface cards, thereby reducing the capabilities of the system or placing a higher load on software.

In almost all cases, conventional applications with fixed QoS constraints simply cannot function correctly in this dynamic environment. As a result, a new class of application called *adaptive applications* has emerged [3, 7]. As their name implies,

such applications *adapt* to QoS changes so that they remain functional in a mobile or changing environment. The TOAST multimedia middleware platform, developed in the Adapt Project [1] and being extended in the Visual Beans Project [25, 13], provides a component-oriented framework that supports the development of adaptive distributed multimedia applications.

This paper is structured as follows. Section 2 outlines the design of the TOAST middleware platform, while section 3 describes how TOAST supports adaptive applications. Section 4 presents some details about the current implementation of TOAST, while section 5 discusses how ongoing research is investigating the application of TOAST to the area of distributed cooperative visualization in the Visual Beans Project.

2. The Design of TOAST

2.1 Introduction

The *Toolkit for Open Adaptive Streaming Technologies* (TOAST) is a CORBA-based multimedia middleware platform. TOAST is an attempt to not only implement multimedia support in CORBA in an RM-ODP [2] inspired manner, but to do this in a way which provides the same plug-and-play component-based mechanism as common multimedia programming frameworks [16, 18, 5]. While basic CORBA multimedia streaming support exists [20], it does not truly follow the RM-ODP model, nor the concepts of component-orientation [24]. A key feature is that TOAST is entirely designed and specified in terms of CORBA IDL, allowing it transcend language and platform boundaries. While this section presents a brief overview of the design of the TOAST platform, a full discussion of the design and implementation of TOAST can be found in [10].

2.2 Component Model

The main abstraction in TOAST is that of a multimedia processing *component*; the basic unit from which applications are built. Examples of components include audio input/output components, video grabbers, network sources/sinks and filters. Following a dataflow or ‘plug-n-play’ model, components exchange continuous media and other data through *stream interfaces* and *flow interfaces*.

Flow interfaces are the fundamental endpoint for media interaction in TOAST and come in two varieties, input and output. Stream interfaces are a direct realization of the RM-ODP concept of a stream interface; an interface which groups one or more flow interfaces into a single composite interaction point, thus simplifying otherwise complicated connection processes. A commonly-used example of a stream interface is a (bi-directional) telephone handset with two unidirectional audio flows. An IDL definition of such a stream interface in TOAST is shown below:

```
interface Handset: TOAST::Stream    interface VideoHandset: Handset
{
    INFLOW audioIn;
    OUTFLOW audioOut;
};
};
```

Components can possess both flow and stream interfaces, allowing more traditional RM-ODP compliance using only stream interfaces, or a more low-level ‘plug-n-play’

multimedia programming framework approach with flow interfaces. An IDL definition of a component with both flow and stream interfaces appears below:

```
interface Phone: TOAST::Component
{
    STREAM handset;
    OUTFLOW recordChannelOut;
};
```

While components may statically declare their stream and flow interfaces as attributes of their main interface (`INFLOW`, `OUTFLOW` and `STREAM` are pre-processor macros), underlying mechanisms allow these to be discovered through a dynamic querying/enumeration process. Also, rather than use enriched IDL to (statically) declare a flow interface's media or information type, this information is always queried dynamically from the flow interface itself.

In addition to stream and flow interfaces, TOAST provides components with event-based interaction. This form of interaction is more asynchronous, dynamic and decoupled than traditional CORBA interaction, and is ideal for the type of application that TOAST is designed for. The programmer can enumerate and query which events a component supports at run-time, and indeed even be notified when new event types are made available by a component. Event-based interaction in TOAST is less ambitious and consequently more lightweight than the CORBA Event Service [19], and is seen as complementary to that service rather than as a replacement.

2.3 Basic Component Interaction

Local binding in TOAST is the process by which two *collocated* stream or flow interfaces are bound together to enable media data to flow between them. Since the local binding of stream interfaces is essentially a compound operation involving the local binding of all compatible pairs of flow interfaces, this section will only examine flow interface local binding.

Flow interface local binding is strictly typed, requiring both flow interfaces to agree on a single, common mediatype which they will exchange. The process of mediatype negotiation is an important one, since without it the programmer must ensure that all connections are meaningful, reducing extensibility and allowing inconsistency. This process of type-checking connections between multimedia processing components is found in DirectShow [5] but not in systems such as MASH [18] or the Berkeley CMT [16] where, for example, a video capture component could be connected to an audio renderer – presumably with painful results.

2.4 Distributed Component Interaction

Whereas collocated flow and stream interfaces can be directly locally bound, to allow distributed interaction requires a *binding object*, a concept defined by the RM-ODP. Binding objects (actually standard TOAST components, albeit complex ones) are essential to allow components to interact over a network, or between incompatible address spaces. Binding objects, or more simply *bindings*, are responsible for the complete end-to-end communications path between the components that they bind.

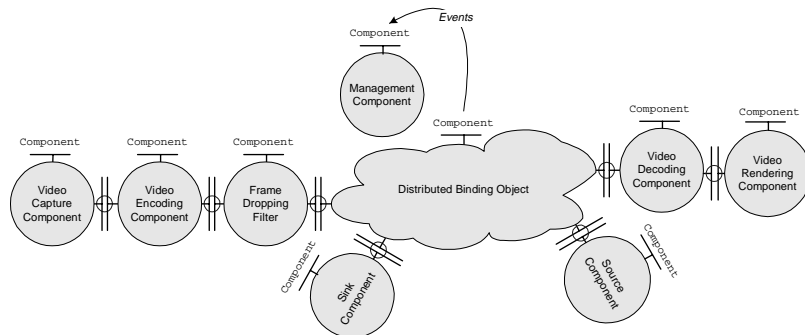


Figure 1: Complex binding with a binding object

Binding objects are created by a *binding factory* and are locally bound to their client components in a semi-recursive fashion. Note that bindings can be arbitrarily complex and are in fact special types of TOAST component. Fig. 1 presents an example of how both local and complex binding is used to build distributed applications.

Binding factories are a crucial part of the TOAST architecture, since they allow components to interact in a distributed manner by creating binding objects to fulfil a particular role. In addition, in combination with the provision of new component implementations via component factories, augmenting the range of available types of binding through the addition of new binding factories is the principal means of achieving extensibility in TOAST.

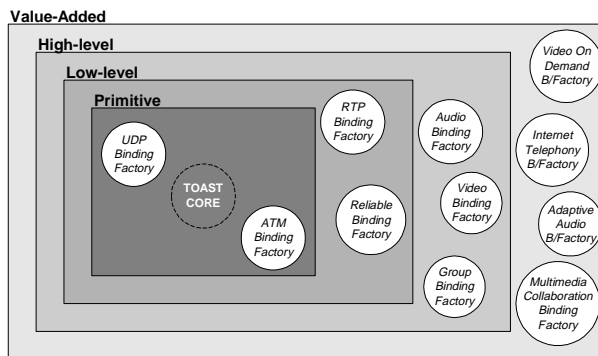


Figure 2: Hierarchy of binding factories for recursive binding

To encourage extensibility, it is intended that binding factories be available to the programmer as a set of distributed services available on the network. Furthermore, the process of creating a binding object is intended to be recursive; whereby higher-level binding factories call upon lower-level binding factories to aid their construction efforts (see Fig. 2). This process of *recursive binding* is explained in [2].

2.5 Component Factories

The dynamic capability discovery features of TOAST components are complemented by component factories which allow component instances to be created dynamically. Each TOAST application or process automatically possesses a component factory

allowing components to be created in that process. A component factory exports a CORBA interface, thus allowing it to be accessed from anywhere in the network (permissions allowing). The TOAST API allows one to create components either in-process, on a given node or collocated with another component, with a single call. In addition, new component implementations may be added to factories at run-time.

2.6 Summary

This section has given a brief outline of the design of the TOAST multimedia middleware platform. In particular the fundamental component model and mechanisms for component interaction have been outlined. The next section now examines how support for adaptation has been built on top of these core abstractions.

3. Support for Adaptation in TOAST

3.1 Introduction

TOAST adopts a framework approach to adaptation, in which specific algorithms and techniques are made possible through a generic support architecture. The benefits of this approach include extensibility with regard to adaptation techniques and mechanisms. This generic support architecture is heavily influenced by the concepts of *open implementation* and *reflection*. This section presents an overview of TOAST support for adaptation; detailed discussion can be found in [10].

3.2 Use of Open Implementation Techniques

3.2.1 Background

Open implementation [14] seeks to overcome the limitations of the ‘black box’ approach to software engineering by opening up key aspects of an implementation to the programmer. Reflection [22] is often used to provide a principled means of open implementation by separating the functionality provided by a system (its interface) from the underlying implementation. In a reflective system, a *meta-interface* allows one to manipulate a causally-connected self-representation of the system’s implementation through *inspection* and *adaptation*. Open implementation and reflection were applied initially to programming languages [14, 26] but have also been used in windowing systems [22], operating systems [29] and CORBA [17, 23].

3.2.2 Open Bindings

TOAST applies reflection and open implementation to supporting adaptation, for example for mobile computing environments. Bindings are responsible for the entire end-to-end communications process, including resource management, signalling, flow filtering/compression and scheduling. To support mobile computing it is crucial that the application be able to exert some control over binding objects [7]. Providing this control via a control interface on a binding has several drawbacks however. In particular, it is difficult if not impossible to design a *general* means of achieving adaptation, mainly due to the sheer proliferation of possible actions, for example in the mobile computing domain [12]. In addition, such an approach would neither be extensible nor dynamic, and would, for example, preclude the use of new adaptation techniques or components introduced during the lifetime of an application or binding.

Instead of a control interface, TOAST allows bindings to offer a meta-interface which operates on a causally-connected self-representation of the implementation of the binding object. This self-representation is modelled as a *component graph*; the nodes of the graph being the TOAST components that make up the binding's implementation; the arcs between them representing the local bindings between these components. Adaptation is achieved by directly inspecting and adapting this self-representation, as will be discussed in section 3.3.

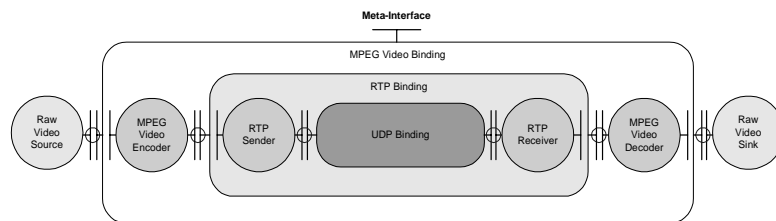


Figure 3: An open binding and its implementation graph

Bindings which expose their implementation in this way are called *open bindings*. An example of an open binding is shown in Fig. 3. Note the use of hierarchical open bindings in the example, where the various subordinate bindings resulting from a recursive binding process can also have their implementations accessed. This recursion is terminated by *primitive components* which do not expose their implementation, for example standard media processing/filtering components or UDP network bindings. TOAST uses *interface mapping* to associate flow interfaces on internal implementation components with a binding's externally visible interfaces in a principled manner.

3.3 Adaptation through Open Implementation

3.3.1 Introduction

As mentioned previously, adaptation is achieved in TOAST by inspecting and adapting the implementation of a binding via its meta-interface; effectively a *procedural* rather than *declarative* approach to QoS management [1]. This approach is the result of earlier work at Lancaster investigating the use of logic or QoS attributes to specify QoS requirements [6]. While valid for certain types of scenario, we have found that declarative techniques do not extend well to the mobile, adaptive environment targeted by TOAST. For example, in the face of greatly varying network connectivity. Achieving adaptation in TOAST is divided into two categories: *minor adaptation* and *major adaptation*, which will both be discussed next.

3.3.2 Minor Adaptation

This form of adaptation is the most basic supported by TOAST. This technique allows fine-grained changes or tweaks to be made to the operation of a binding while it is running. The intended use is to allow the programmer or adaptive agent (e.g. a control script or active object) to counteract slight changes in the environment, such as network QoS fluctuations or end-system resource availability.

The first step in minor adaptation is the identification and location of the appropriate components within a binding's implementation graph using the binding's meta-

interface. Once located, these components may be interrogated, for example to determine the true state of QoS or performance. If desired, adaptation can then be achieved by accessing these components' control interfaces to alter their operation. An example of minor adaptation would be locating a primitive network binding component offered network QoS reports via its interface (e.g. through TOAST events), and using this information to change the encoding rate/quality of a video encoder component elsewhere in the implementation graph.

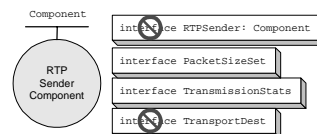


Figure 4: Restricting access to implementation-graph components

While such techniques can be used to good effect, it is also important to allow a meta-interface to maintain consistency in the face of adaptation. For example, one could easily access a network source component in a binding and change its destination address, thereby destroying the operation of the binding. For this reason, the meta-interface and implementation component graph refer to components using opaque IDs; to access a component's interface one must request it specifically. This allows a meta-interface to restrict access to certain 'safe' components, or, by way of TOAST COM-style multiple interface support (see [10]), even to only certain interfaces on certain components (see Fig. 4).

3.3.3 Major Adaptation

The second, most drastic, form of adaptation supported by TOAST is major adaptation. Similarly to minor adaptation, one uses a binding's meta-interface to inspect its implementation; however major adaptation involves changing the structure of this implementation graph to alter the implementation of the binding. Since a binding's implementation is represented as a graph, the meta-interface provides methods to add and remove nodes and arcs (components and local bindings) to and from this graph. In addition, a number of higher-level or *compound* adaptation operations are provided. These include operations allowing the transparent hot-swapping of one side of a local binding and completely replacing one component with another 'similar' one. A full discussion of major adaptation including the meta-interface API appears in [10].

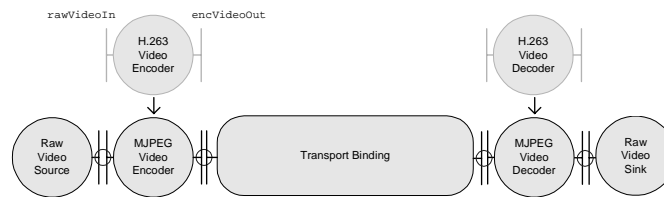


Figure 5: Using major adaptation to reconfigure a binding

Perhaps the most common use for major adaptation is to alter the 'big-picture' strategy of a binding to adapt to an environmental change. This might involve replacing unsuitable components with newer ones, the insertion of extra components

(e.g. QoS filters [28]) or the removal of redundant components. An example of major adaptation is shown in Fig. 5. Here, the meta-interface is being used to adapt a unidirectional video binding in the face of a network change from ~2Mbit/s WaveLAN to a <9.6Kbit/s GSM dialup link. In this case, the video compression type must be changed to match; consequently the motion-JPEG CODEC components are replaced with H.263 counterparts to provide a much lower-bitrate video channel.

Perhaps more so than with minor adaptation, the maintenance of consistency is extremely important with major adaptation, since once could quite easily remove all components from a binding's implementation, or alternatively break all local bindings, thereby preventing correct operation. Again, the meta-interface is responsible for consistency, and for determining, and vetoing, all major adaptation operations. The provision of compound adaptation operations, such as replacing components, is a direct result of lessons learned in this area; while the act of inserting/removing a filter component would be deemed acceptable by a meta-interface, the individual breaking/binding operations that make up this compound operation would not be. The compound operations therefore allow adaptation operations that would otherwise be unacceptable to the meta-interface.

3.4 Summary

This section has outlined how the TOAST middleware platform supports adaptation and reconfiguration through its use of open implementation and reflection techniques. The use of a component graph structure and meta-interface to support both major and minor adaptation in a principled and intuitive manner has been described. It should be noted that, since TOAST adaptation support is not only inherent but also built on top of the existing TOAST component model, *all* applications written using TOAST can conceivably be adapted. In addition, the dynamic nature of the TOAST component model allows new components to be merged into an existing application using the adaptation mechanisms.

4. Current Implementation

At present, TOAST has been implemented in C++ for the Win32 platform and Java for the JDK1.3 platform using the ORBacus [21] C++/Java CORBA system. A growing library of components exists for both C++ and Java, including video and audio capture, processing and playback. A number of demonstration applications showing the adaptive aspects of TOAST have been designed and implemented. Currently, ongoing research is investigating the uses for TOAST in the area of distributed co-operative visualisation (see Section 5).

To demonstrate that the design and specification of TOAST in CORBA does not introduce overhead making real-time media processing impractical, the remainder of this section presents some measurements from the current implementation¹. Note that more detailed measurements, as well as a qualitative evaluation of TOAST, appear in [10].

¹ All measurements were taken from the C++/Win32/ORBacus3.1 implementation of TOAST running on an Intel Pentium II 350MHz PC with 128Mb memory running Windows NT4.0 SP5.

Perhaps the simplest measurement of TOAST is the throughput, in terms of media data buffers per second, of a TOAST local binding between two flow interfaces in the same process. Compared to a C++ virtual function call which achieves about 11 million buffers (i.e. invocations) per second, a local binding achieves roughly half that figure at about 5.7 million per second. While TOAST obviously introduces significant overhead, the throughput is sufficient for almost all common media processing tasks; an example of the throughput in terms of buffers per second for some common media types is shown in Table 1.

Media Type	Throughput Required
Full Motion Video – all types (1 frame per buffer)	24, 25 or 30 bufs/sec
CD Quality Digital PCM Audio (1k buffers)	172 buffers/sec
CD Quality Digital PCM Audio (512 byte buffers)	345 buffers/sec
Telephone Quality PCM Audio (1k buffers)	16 buffers/sec
GSM Compressed Audio (1 frame per buffer)	50 buffers/sec
10Mbit/s Continuous Media (1k buffers)	10,240 buffers/sec

Table 1: Example media throughput requirements

To illustrate the overheads imposed by the TOAST component model, Figure 7 presents the throughput of *pipelines* of successive sizes made up of filter-style components. Two sets of measurements are presented; one for pipelines of non-threaded components, the other for pipelines made up of components which each have a worker thread handling their (null in this test case) processing. Contrasting these figures with raw local binding throughput, it is apparent that though throughput is greatly reduced by the overhead of the component model, it is still more than enough for most purposes. An interesting observation from Fig. 7 is that of the overhead imposed by multithreaded components; the context-switching and other thread synchronisation having a large impact on throughput values.

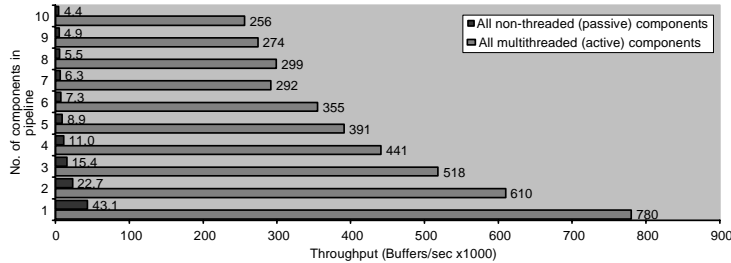


Figure 7: In-process component pipeline throughput

5. Application to Distributed Cooperative Visualization

The open adaptive streaming technology in TOAST is designed to support distributed multimedia applications and current work (within the Visual Beans project [25]) is investigating this by studying one particular application, *distributed cooperative visualization* (DCV). Visualization is a key technology for many scientific, engineering, medical, geographical and social science applications and it is common for datasets to be large or complex or both. DCV aims to extend this so that groups of professionals can work at a distance, sharing visualization results, procedures and activities [4]. Various terminology has been used. *Distance visualization* [11] presents

visualization on the Grid using dispersed resources and supporting dispersed end users. *Teleimmersion* [15] immerses a group of dispersed users in a single virtual world, which contains not only the focus of their common task but also virtual representatives of each user, referred to as avatars. In some sense the participants are not only looking at the focus of interest but are within it.

Experience of DCV [8, 27, 4] has revealed a number of directions and problems that require investigation. We outline here the ones that relate to component technology and TOAST in particular.

- Allowing the participants to access the visualization software actively, by contrast with previously prepared images on a distributed whiteboard, enables alternative plans to be tested (“What happens if I change this parameter?”) and seen and optionally controlled by all. Different experts in a collaboration can offer expertise on different data sources, algorithms and phenomena.
- Remote data archives or data archives local to one user need to be accessible to the collaboration. Although, for reasons of quantity and sensitivity, some data processing takes place close logically close to the data, it is still necessary to accommodate large data transfer to all participants.
- Video communication of each other adds to the level of awareness of all concerned, but a sufficiently good quality of voice communication is critical to the collaboration.
- Availability of Grid technology means that computational resources and data access will be dispersed without the location being known to the participants. A coherent local visualization environment is required.
- The use of the visualization system can itself result in the delivery of a movie to all participants. A good example is the study of time-varying phenomena. While these are being studied, explanation and discussion may continue. It can be important to know that the participants are looking at the same frame at the same time. Whether an action replay is under the control of one person or controlled on an individual basis, synchronization may well be vital.
- Thus several types of information are required (for example voice, video, whiteboard, image, data, movie, web link), some of which each require a continuous flow of information.
- Some visualization processes may require intensive use of storage, computational or networking resources, which may therefore compete with the processing of continuous flows.
- A collaboration will involve a number of host computers of diverse power and network connectivity. Developments in mobile technology will increase the range of QoS changes to which the collaborative application will need to adapt.

These can be summarised in the following major issues.

- The need for resource management to control the competing demands of several continuous flows, transmission of data and low-latency distributed interaction.
- The need for adaptation to diverse and changing resources.

The Visual Beans project [25] is investigating the open adaptive streaming approach provided by TOAST to manage these problems. By merging a component based visualization system (specifically VisAD) [13], into the TOAST component model,

visualization applications gain access to the adaptation and distribution support offered by TOAST. This approach allows visualization systems to benefit from existing TOAST continuous media support (e.g. pre-existing components) as well as providing an integrated approach to resource management.

6. Conclusions

This paper has outlined the design of the TOAST distributed multimedia middleware platform, in particular its component-oriented approach to multimedia systems. The architectural support for adaptation offered by TOAST was then described, along with some details of current implementation work. Finally, ongoing research was introduced that is using TOAST to investigate its application to distributed cooperative visualization.

Acknowledgements

The authors acknowledge the support to the Visual Beans Project provided by the (UK) Engineering and Physical Sciences Research Council (EPSRC) under grants GR/M81779 and GR/M82011.

References

1. Blair, G., Coulson, G., Davies, N., Robin, P. and Fitzpatrick, T., "Adaptive Middleware for Mobile Multimedia Applications", Proc. NOSSDAV'97, Missouri, USA, May 1997.
2. Blair, G. and Stefani, J., "Open Distributed Processing and Multimedia", Addison-Wesley, Harlow, England 1998.
3. Braden, R., Clark, D. and Shenker, S., "Integrated Services in the Internet Environment", IETF Request For Comments, RFC 1633, June 1994. <http://www.ietf.org/rfc.html>
4. Brodli, K.W., Duce, D.A., Gallop, J.R. and Wood, J.D., "Distributed Cooperative Visualization", Eurographics '98 State of the Art Reports, deSousa, A.A. and Hopgood, F.R.A. (Eds), pp27-60, 1998.
5. Chatterjee, A. and Maltz, A., "Microsoft DirectShow: A new media architecture", SMPTE Journal, December 1997, Vol.106, No.12, pp.865-871.
6. Coulson, G and Waddington, D.G., "A CORBA-compliant Real-time Multimedia Platform for Broadband Networks", Proc. TRENDS'96, Aachen, Germany, September 1996.
7. Davies, N., Friday, A., Blair, G. 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.
8. Duce, D.A., Gallop, J.R., Johnson, I.J., Robinson, K., Seelig, C.D., Cooper, C.S. : Distributed Cooperative Visualisation - Experiences and Issues from the MANICORAL Project, in *Proceedings of the Eurographics Workshop on Visualisation in Scientific Computing*, Eurographics Association, 1998.
9. Fitzpatrick, T., Blair, G., Coulson, G., Davies, N. and Robin, P., "A Software Architecture For Adaptive Distributed Multimedia Systems", IEE Proceedings – Software, Special Issue on Configurable Distributed Systems, 1998.
10. Fitzpatrick, T., "Open Component Oriented Multimedia Middleware for Adaptive Distributed Applications", PhD thesis, Lancaster University, May 2000.
11. Foster, I., Stevens, R. : Corridor One, An Integrated Distance Visualization Environment for SSI and ASCI Applications, DoE NGI Testbed Workshop, July 1999, <http://www.itg.lbl.gov/NGI/Jul99/IFoster/index.htm>

12. Friday, A., "Infrastructure Support for Adaptive Mobile Applications", Ph.D. Thesis, Computing Department, Lancaster University, Lancaster, England, September 1996.
13. Gallop, J., Cooper, C., Johnson, I., Duce, D., Blair, B., Coulson, G., and Fitzpatrick, T., "Structuring for Extensibility - Adapting the Past to Fit the Future", In Slagter, R.J., ter Hofte, G.H., and Stiemerling, O., (Eds.), *Proceedings of CBG2000, the CSCW2000 workshop on Component-Based Groupware, December 2, 2000, Philadelphia, USA*, Telematica Instituut, The Netherlands, ISBN 90-75176-24-4.
14. Kiczales, G., Des Rivieres, J. and Bobrow, D., "The Art of the Metaobject Protocol", MIT Press, 1991.
15. Leigh, J., Johnson, A.E., Brown, M., Sandin, D.J., DeFanti, T.A.: Visualization in Teleimmersive Environments, *Computer*, Vol 32, No 12, December 1999.
16. Mayer-Patel, K. and Rowe, L., "Design and Performance of the Berkeley Continuous Media Toolkit", in *Multimedia Computing and Networking 1997*, Freeman, Jardtetzky and Vin (Eds), Proc. SPIE 3020, pp 194-206, 1997.
17. McAffer, J., "Meta-Level Architecture Support for Distributed Objects", Proc. Reflection'96, G. Kiczales (Ed.), pp39-62, San Francisco, USA, 1996.
18. 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., Wu, D. and Smith, B., "Towards a Common Infrastructure for Multimedia-Networking Middleware", Proc. NOSSDAV'97, Missouri, US, 1997.
19. Object Management Group, "CORBA Event Management Service", OMG Document formal/97-12-11, Object Management Group, Framingham, MA, USA.
20. Object Management Group, "Control and Management of Audio/Video Streams", OMG Document formal/98-06-05, Object Management Group, Framingham, MA, USA.
21. Object Oriented Concepts Inc., "ORBacus for C++ and Java", Object Oriented Concepts Inc, Billerica, MA USA. Available on the Internet at <http://www.ooc.com/ob/>
22. Rao, R., "Implementational Reflection in Silica", Proc. ECOOP'91, Lecture Notes In Computer Science, P. America (Ed.), pp251-267, Springer-Verlag, 1991.
23. 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, Jyvaskyla, Finland, 1997.
24. Szyperki, C., "Component Software: Beyond Object-Oriented Programming", Addison-Wesley, Harlow, England 1998.
25. The Visual Beans Project web page: <http://www.acu.rl.ac.uk/VisualBeans/>
26. Watanabe, T. and Yonezawam A., "Reflection in an Object-Oriented Concurrent Language", Proc. OOPSLA'88, Vol. 23 of ACM SIGPLAN Notices, ACM Press, 1988.
27. Wood, J., Wright, H., Brodlie, K. : Collaborative Visualization, Proceedings of IEEE Visualization 97, pp 253-259, ACM Press, 1997
28. Yeadon, N., "QoS Filtering for Multipeer Communications", PhD Thesis, Lancaster University, September 1996.
29. Yokote, Y., "The Apertos Reflective Operating System: The Concept and Its Implementation", Proc. OOPSLA'92, Vol. 28 of ACM SIGPLAN Notices, pp414-434, ACM Press, 1992.