CORBA Lacks Venom

Kevin Curran, Gerard Parr
Telecommunications & Distributed Systems Research Group
Northern Ireland Knowledge Engineering Laboratory
University of Ulster, Coleraine Campus, Northern Ireland, UK
Email: kj.curran@ulst.ac.uk

Abstract

Distributed objects bring to distributed computing such desirable properties of modularisation, abstraction and reuse easing the burden of development and maintenance by diminishing the gap between implementation and real-world objects. Distributed objects, however, need a consistent framework in which inter-object communication may take place. The Common Object Request Broker Architecture (CORBA) is a distributed object standard. CORBA's primary protocol is the Internet Interoperable Object Protocol limited to blocked synchronous remote procedure calls, over TCP/IP which is inappropriate for systems requiring timely guarantees.

The Real-time Wide Area Network Dissemination Architecture Protocol (RWANDA) overcomes the synchronous limitation by building on top of an asynchronous group communication model where applications only pay for required quality of service (QoS) such as multicast, virtual synchrony and encrypted communication. In RWANDA, information sources use channels to disseminate information to a potentially large and changing set of channel subscribers. RWANDA recognises the differing media characteristics and transport requirements of multimedia by providing a protocol composition framework that extends to incorporate yet unsupported communication protocols, qualities of service and optimised multimedia stacks. RWANDA provides an asynchronous foundation necessary for developing a large scale wide area network continuous media protocol.

Key words: Distributed systems, multimedia, object frameworks, middleware, continuous media

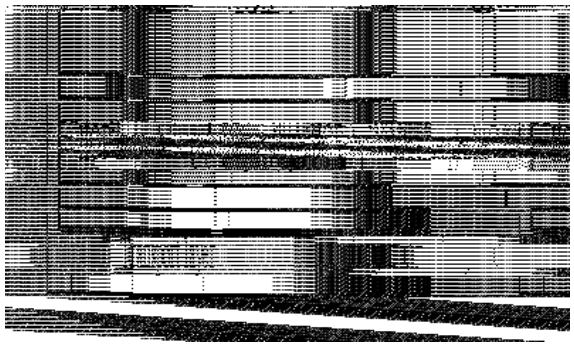
1 Introduction

The Object Management Group (OMG) is a consortium of more than 300 hardware, software and end-user companies. Some of the principal OMG partners are DEC, with its Objectbroker product, IONA Technologies Ltd. (Dublin, Ireland), with its Orbix implementation, and IBM, with its Distributed System Object Model (SOM/DSOM) [Tibbitts95]. These companies along with ObjectDesign, were authors of the Common Object Request Broker Architecture (CORBA) [OMG95]. CORBA specifies the architecture of an Object Request Broker (ORB). The ORB is defined to enable and regulate interoperability between objects and applications being part of a larger vision called the Object Management Architecture (OMA).

The CORBA specification does not address implementation details and leaves many areas undefined. This unfortunately resulted in proprietary technologies used by the various CORBA vendors, which contrasts with the single Microsoft OLE specification and implementation [Betz94].

1.1 CORBA

The Common Object Request Broker Architecture , introduced in 1991, is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA defines true interoperability by specifying how Object Request Broker's from different vendors can interoperate. It defines the Interface Definition Language (IDL) and the Application Programming Interfaces that enable client/server object interaction within a specific implementation of an ORB.



More sophisticated recovery mechanisms allow a blocked procedure call to be redirected to an alternative server. But if a client process has to be re-routed via an RPC, application performance may suffer. Not all RPCs are strictly synchronous. A few support some form of asynchronous communications, which typically involves a nonblocking function call. But these RPCs are more the exception than the rule and can be exceedingly difficult to implement. In addition, because most RPC-based systems have no way of establishing peer-to-peer relationships, they are not well suited to object-oriented programming. Implementing a distributed object-oriented environment using RPC development tools demands resources and involves significant application overhead.

Peer-to-peer protocols do not possess this limitation, but they are not without drawbacks. Once the client has passed a message along to the server, it is free to go about its business. Unfortunately, it's also free to pass along more messages. It is easy to see how (under the wrong circumstances) peer-to-peer protocols can drive up traffic volumes exponentially. Ironically, the master-slave relationship that typifies request-reply protocols also guards against this scenario. As long as the client is waiting to hear from the server, it is unable to do anything else.

Ultimately, it's not a question of one communications protocol being better than another. Both can increase the strain on a network (under different conditions) and both must be treated with caution.

1.3 Generic Transport Stack Limitations

Multimedia is composed of varying types such as audio, video, plain text, control information etc. and within these types, there exists a multitude of formats such as JPEG, MPG etc. Therefore to use the same protocol stacks to cater for all these transport types is not the ideal scenario.

Transport protocols such as CORBA's IIOP, Java's RMI and DCOM all transport the varying media types through the same protocol stack. Therefore if a video file is transported through the same connection as an audio file, the video data will have to adopt the packet size allocated to the audio file. Audio in general runs more efficiently with smaller packet sizes. Isochronous Multimedia traffic can tolerate some loss, the problem is that data which misses it's expected delivery time is of no use. Therefore it is more efficient to lose smaller packets than larger packets. However, increased packets means increased header processing in routers. Small packet sizes are not optimal for graphical data.

The ideal transport protocol for dissemination-oriented communication provides a basic service that supports multicast streams, with incremental extensions and specialisation's to support conversational and request-response communication as part of the same base protocol mechanism [Cheriton95].

Traditional transport protocols utilise an identical stack for all the media. A more efficient method would construct optimised protocol stacks for each of the media e.g. audio, text, video. Maximum benefit would be achieved if this could be implemented at run-time to cater for the applications particular preferences.

1.4 CORBA Limitations

The CORBA model is lacking for the class of systems requiring real-time guarantees [Cingser96]. It is fundamentally based on a blocked synchronous RPC model [Resnick96], rather than an asynchronous Message-oriented middleware (MOM) model, which hinders the creation of real-time systems.

Currently most real-time CORBA systems are implemented on UDP/IP connectionless protocols, acting as a "plumbing" approach to the rest of the system which is structured on top of CORBA protocol.

In the case of CORBA, one might suggest basing the multimedia data transport work on top of the IIOP protocol. There is however a peculiar difference that must be noticed when developing, for example, a real motion video application and a traditional networked one: building video applications on top of inherently request/reply computational model like CORBA, inevitably requires implementing the audio and video delivery outside CORBA requests. The ORB based model suits perfectly well in most cases when searching databases, managing objects and performing tasks on a

request/reply basis is needed, but modern video applications require much more than just this kind of behaviour. They need streaming capabilities and sensible Quality of Service in the delivery of data. It is a generally well-known fact that multimedia systems require substantial support from the underlying transport and signalling protocols to provide adequate QoS [OrbTalk97].

The IIOP protocol doesn't currently equip CORBA with any streaming capabilities.That said, one must openly admit that, without introducing proprietary extensions to CORBA, one cannot do without the so-called "plumbing approach" i.e. handling audio/video transmission on a different network protocol layer. There is currently no standard protocol on the Internet, providing multimedia programmers with the ability to control and negotiate the quality of service of a given stream of multimedia data.

One may build real-time requests using one-way CORBA operations. However, some CORBA implementations seem to execute some subsequent one-way operations in a LIFO fashion [RMI97]. These resulted in the need to build yet another kind of "sequencing protocol" and introduced considerable jitter into the video stream, confirming the notion that applying the CORBA request/reply model is not appropriate when streaming time-dependent data transfer is required.

CORBA calls are not asynchronous in reality [Resnick96], since the CORBA specification clearly permits the ORB to even block while sending a one-way call. Such behaviour is highly undesirable when high speed video is to be transmitted and displayed.

Another problem within real-time CORBA systems is the lack of facilities to cope with notification to multiple subscribers of state changes in objects. This is due to the design of the CORBA event framework. However this may be overcome through building atop IP multicast. Applications built with static compile-time interfaces with these object models, are just as monolithic and inflexible as local address-space C++ program's [Schmidt97], appearing internally as 'object' systems to the developer but not externally to the user.

Present-day ORB's limit the number of object invocations per process to a relatively low 500-1000 calls per second [Maffeis97]. What if an object is to be distributed to 20 workstations, has 50 attributes, and changes once per second? CORBA doesn't yet have a standard for doing this, but the OMG will allow passing of objects by value in CORBA 3.0 [Mosj94].

It has been stated that this is using 'today's technology to build yesterday's applications' [Cingiser96]. For this community of users, who desire to build static compile-time applications that access existing databases, CORBA is a worthwhile approach. It provides an object oriented model that maps well to network programming abstractions e.g. Object interfaces. It is somewhat akin to programming with Remote Procedure Calls only at a higher level.

CORBA seems likely to continue to expand in terms of end-users, nonetheless there needs to be a shift in the foundational infrastructure before it is capable of supporting large scale isochronous multimedia applications.

2 The RWANDA Protocol

The RWANDA protocol has arisen to cater for the category of large-scale isochronous wide area multimedia applications. RWANDA builds upon the iBus framework [Maffeis97]. We have stated the lack of timely guarantees, scalability, object abstractions and inefficient transport protocols offered to a greater or lesser degree by CORBA. The RWANDA protocol is outlined below and the implementation of a real-time multimedia dissemination protocol is outlined with specific mention of the problems that RWANDA helps to overcome in an application of this nature.

2.1 Protocol Elements

The protocol elements (objects) consist of basically of four components. These are :

1. Channel 2. Talker application 3. Listener application 4. Posting 5. Filter

2.1.1 Channels

Channels are a multicast medium into which taker applications push objects, and to which listener applications can subscribe to receive those objects.

A channel maps into an IP multicast group or a point-to-point UDP connection. Uniform resource locators are used for naming channels. These are denoted by a URL such as `Videoplayer://227.134.3.63/videos/indy`

2.1.2 Talker Applications

These are Java applications that push or pull Java objects via one or more channels.

2.1.3 Listener Applications

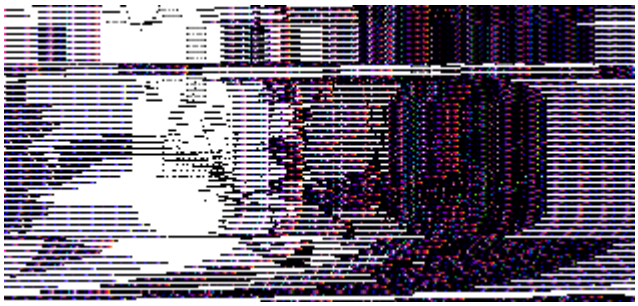
These are Java applications that subscribe to one or more channels to receive Java objects such as the recipients of a video conference.

2.1.4 Posting

These are serializable Java objects sent through channels. This is the actual data to be transmitted such as the video, audio or whiteboard text.

2.1.5 Filter

Filters extract the different media types from within the application and create suitable run-time protocol stacks to enable streamlined transport communication. Filters are responsible for re-assembling the flows at the receiver.

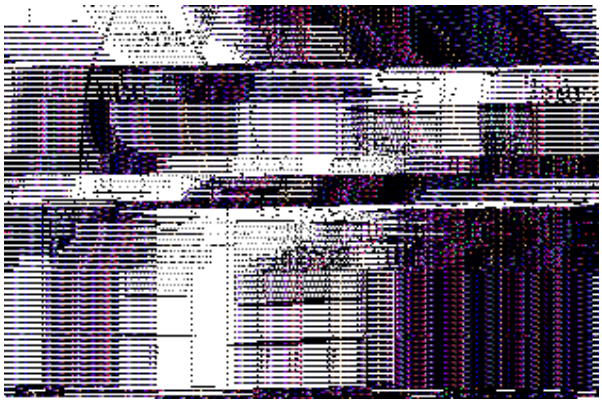


framework allows protocol stacks to be composed dynamically (e.g. at run-time). This allows for a flexible architecture suited to client application needs.

2.3 Mixed Media eXtraction - MMX

Multimedia more so than other applications is recognised as containing distinct media formats deserving of distinct transportation treatment. As already mentioned, audio packets can be shipped much smaller than video packets. We are concerned with timeliness rather than reliability. Depending on whether compression is present or not we can also vary the size of audio and video packets. Text media in the context of files or control data associated with a video conference requires acknowledgements – as provided by the TCP protocol.

Traditional frameworks push these varying types through identical protocol stacks. No optimisation for each media type is performed.



3 Conclusion

The design of and development of a distributed application is quite complex. The development of these applications can be simplified using distributed objects models. These models typically provide interface description languages, or strong object oriented reference frameworks which allow the developer to concentrate exclusively on the development of server's objects and customer invocations of methods without worrying about network communication. The server's objects interfaces are clearly defined, and as a result the final development becomes easier. CORBA promises much in the creation of interoperable object, however, it falls short when it comes to ease of programming, costs, object orientation features and real-time guarantees. It provides suitable resources for the creation of non real-time applications.

RWANDA is built upon a Java middleware application, closely modelled on the iBus framework, supporting event driven applications on top of group communication protocols implements a quality of service framework that allows programmers to compose protocol stacks for unreliable communication, reliable multicast, virtual synchrony, message encryption, and so forth. This provides a framework where applications need only pay for quality of services they need. iBus also allows the dynamic creation of protocol stacks which brings an increased flexibility to network programming allowing stacks to be composed of relevant functionality. Channels allows for a large scale groups of receivers to receive information. The channels also allows heterogeneous receivers with differing capabilities to receive information.

RWANDA's recognises the differing media characteristics and transport requirements within multimedia and provides run-time composable protocol stacks. This allows the tailoring of specific stacks to cater for the varying media. The run-time property allows for a flexible development environment, delaying the necessity for defining protocols until the latest possible stage thus allowing extension's of media to the architecture. Traditional methods do not have this flexibility. RWANDA provides the multicast group process abstractions necessary upon which a Large Scale wide area network real-time multimedia protocol may be developed.

References

[Betz94], Betz, Mark, "Interoperable Objects", Dr. Dobb's , 1994.

[Birman96] Birman, Kenneth P., Building reliable and secure distributed systems. Manning Publications Co., US. 1996

[Cheriton95] Cheriton, David. Dissemination-Oriented Communication Systems, Stanford University, Tech Report, 1995

[Cingser96] Cingser, Lisa., Dipippo, Roman., Expressing and enforcing timing constraints in a CORBA environment., 1996.

[Maffeis97] Maffeis, Silvano. iBus - The Java Intranet Software Bus, http://www.olsen.ch , 1997.

[OMG95] The Common Object Request Broker: Architecture and Specification, OMG, http://www.omg.org, July 1995

[Resnick96] Resnick, Ron. Toward the Integration of WWW and Distributed Object Technology: Distributed Objects on the WWW. *OPSLA'96* http://www.eng.uci.edu/~peilei/index.html , *1996.*

[RFC 1644] Braden, R. T., T/TCP-TCP Extensions for Transactions, Functional Specification, 38 pages, July 1994.

[RFC 1819] Deering, S. Internet Stream Protocol Version 2 (ST2) Protocol Specification, August 1995

[RMI97] RMI and IIOP FAQ, http://www.javasoft.com/pr/1997/ june/statement970626-01.faq.html , June 1997

[Schmidt97] Schmidt, D., Levine, D. Harrison, T. An ORB endsystem architecture for hard real-time scheduling. *Submitted to OMG in response to RFI ORBOS/96-09-02*, Feb 1997.

[Tibbitts95] Tibbitts, Fred, "CORBA: A Common Touch for Distributed Applications", Data Communications, May 1995