



The ATON Project

Center for Research in Electronic Art Technology
University of California, Santa Barbara
Computer Vision and Robotics Research Laboratory
University of California, San Diego
CalTrans Test-Bed Center For Interoperability



ATON Report 2001.06.6: Support for Streaming HPDM Applications and the CORBA A/V Standard

Stephen Travis Pope, Frode Holm, Ahmi Wolf, and Francisco Iovino
{stp, frode, ahmi, francisco}@create.ucsb.edu
CREATE, UCSB, June, 2001

Contents

- Preface2
- Introduction2
- Context and Related Work3
- Requirements of HPDM Applications3
- Distributed Object Computing Frameworks4
- Recent Extensions to the OMG CORBA Specification5
- The Real-Time CORBA Specification5
 - Discussion6
- The CORBA A/V Streaming Specification7
 - Discussion9
- Performance of DOC Frameworks10
- Options for Deploying HPDM Applications11
- Conclusions11
- References11

Preface

Within the DiMI ATON Project's Research Thrust 5.1, we at UCSB have been working on software to support what we call *High-Performance Distributed Multimedia* (HPDM) software systems. For these systems, we need to be able to design, construct, and manage multi-server real-time applications, and we require high-level management tools that we call the *Distributed Processing Environment* (DPE). This document describes our motivations, requirements, and applications, and compares several of the options available for building state-of-the-art HPDM software. We discuss the application developer's requirements for HPDM software frameworks, and the needs of the DPE tool set. We will survey the industry standards related to this field, and compare them to our requirements. We devote specific attention to the recent Object Management Group standards for real-time CORBA and the CORBA Audio/Video streaming service.

Introduction

Software to support Distributed Object Computing (DOC) has been under development for over 15 years. The Object Management Group (OMG, www.omg.org) has produced several versions of the Common Object Request Broker Architecture (CORBA), and a number of corporations support various versions of CORBA in their Object Request Broker (ORB) products. Enhancing and extending CORBA is also the topic of a large research literature.

In previous projects with the CalTrans TCFI, researchers at CREATE have investigated the real-world interoperability of different CORBA ORB implementations, as well as their support for multi-platform, multi-language applications (see the references: IDOT and DRIVE projects). It has been our experience that the basic CORBA version 2.2/2.3 functions are indeed widely supported across ORBs, platforms, and languages, but that the availability of higher-level (and newer) versions of CORBA (2.4.X and 3.0) and optional CORBA services (e.g., CORBA Messaging and CORBA A/V) still differs widely among ORB implementations.

Within the ATON project, we were tasked with developing a framework for wide-area shared virtual environments that support multimedia control and content streaming among different kinds of applications (e.g., user tracking, robot control, video streaming, and spatialised audio). The HPDM/DPE design document (ATON/UCSB Report 2000.06.1) described the requirements of our ATON applications in terms of the language and object models, the development libraries and tools, the run-time infrastructure, and the end-to-end application performance (see Table 1 in the reference).

Given these requirements, it was obvious that we should investigate two recent extensions to the CORBA specification: the Real-Time CORBA standard (OMG 2001c) the CORBA Audio/Video Streaming Service (OMG 2000), both of which were available as draft standards at the start of the project. Based on the design requirements of the ATON project, and on the status, stability, and performance of implementations of these two CORBA extensions, we decided that we should base our HPDM support libraries and DPE tools on standard off-the-shelf CORBA 2.3-level ORBs, and that we must build the infrastructure for managing application-level Quality-of-Service (QoS) as well as the DPE tools ourselves.

Context and Related Work

The requirements of the ATON Project, and of HPDM applications in general (see the following section), determined our approach to the task of tool selection for the real-time distributed infrastructure we use. The research literature related to such systems is indeed diverse, as is the set of real-time applications under investigation in other groups.

The “high-end” research in the field (e.g., O’Ryan et al., 2000; Feng, Syyid, and Liu, 1997) addresses “hard real-time” applications such as avionics and medical systems, and often rejects the use of such “untrustworthy” protocols as TCP/IP, as well as the use of general-purpose operating systems, opting instead for native ATM protocols (e.g., IIOP over ATM), and special-purpose real-time operating systems such as LynxOS or VxWorks. These applications often require tuned real-time (possibly embedded) ORBs, predetermined schedules and scheduling policies, and even special-purpose hardware.

Several alternative (light-weight) approaches have concentrated on building “best-effort” real-time systems using off-the-shelf ORBs and operating systems. The two systems that most closely address our kinds of problems are the BBN Quality Objects (QuO) system (Zinky, Bakken, and Schantz, 1997) and the UCSB Realize system (Kalogeraki, Melliar-Smith, and Moser, 1997). These systems share the two features of (1) run-time system monitoring, performance tuning, and reconfiguration (with possible object migration), and (2) special QoS definition languages (QDLs) that are used by the run-time scheduler/manager in parallel with, but separate from, the operation of ORBs and the CORBA IDL interfaces of the objects they manage. As we shall present and discuss below, we have opted for an architecture that more closely corresponds to this approach than to the “hard real-time” solutions of the first kind.

Requirements of HPDM Applications

The design document that we prepared in the first phase of the ATON project (ATON/UCSB Report 2000.06.1) gave several examples of HPDM applications, and summarized the requirements that we feel must be satisfied for a flexible and powerful HPDM development framework. We described the features of the class of HPDM applications that we wish to support; these result in requirements such as:

- “soft” real-time performance (on the order of 2-5 msec typical latency and 1-2 msec of latency jitter, e.g., usable for event streaming using the MIDI protocol or for user-tracking devices) on off-the-shelf hardware, operating systems, and ORBs;
- streaming of low-resolution (or low-frame-rate) video and/or 2-8 channels of CD-quality audio (400-800 kByte/sec throughput)—possibly out-of-band in terms of the CORBA data exchange protocols;
- ability to respond to unpredictable event bursts, as often seen in multi-user virtual environments and musical performance systems (i.e., static scheduling is not possible);
- 1-to-1 synchronous (blocking) message-passing and 1-to-many asynchronous (though still guaranteed) “event” distribution;
- standard CORBA services such as naming and (optionally) trading; and
- QoS description and run-time enforcement.

Part of our effort has been to develop a prototype of a management system for HPDM applications, which we call the Distributed Processing Environment (DPE). This framework has to provide a set of software tools for deploying, starting, monitoring HPDM applications, including:

- flexible description of configuration “scenarios” of multi-server HPDM applications;
- automatic start-up and controlled shut-down of complex HPDM applications;
- run-time monitoring of server performance and resource (CPU, memory, network bandwidth) usage, and some form of load-balancing; and
- fault-tolerance and server re-start upon (service, ORB, or platform) failure.

The basic infrastructure and programming environment requirements for DPE and HPDM applications include support for:

- the use of off-the-shelf hardware and operating systems (we use Linux, Solaris, IRIX, MS-WindowsNT, and MacOS);
- the use of several development languages (we use C, C++, Java, and Smalltalk);
- several kinds of data interchange, including remote synchronous 1-to-1 message-passing, a synchronous 1-to-many message-passing (AKA events or messaging), and multimedia data streaming (of control information or media content);
- relative ease of learning and use (appropriate for a CS graduate student to become proficient with in four weeks or less); and
- the ability to describe and use application-level quality-of-service (QoS) requirements.

The issues listed above have influenced the design and implementation of the HPDM infrastructure and the DPE tools. We will discuss the design alternatives below, and compare the options that we evaluated for satisfying these requirements.

Distributed Object Computing Frameworks

Any distributed programming library must include at least two components: (1) a standardized external data representation (XDR, which describes how data is to be passed over the network in a machine-independent fashion), and (2) a standardized format for 1-to-1 (client-to-server) remote procedure calls (RPC) or (in the case of object-oriented programming languages) remote method invocations (RMI). Distributed programming packages often provide a format for describing XDR/RMI interfaces in an implementation-language-independent way, for example, using a special interface definition language (IDL).

Two other desirable features of distributed programming libraries for multimedia applications are: (3) a mechanism for asynchronous 1-to-many communication among components (also called event distribution, notification, publish/subscribe services, the dependency mechanism, or signals), and (4) a method of specifying an application's quality-of-service (QoS) requirements. in terms of network delays (latency) and “burstiness” (latency jitter).

Typical distributed programming frameworks (e.g., CORBA, COM, or JavaRMI) consist of a user program (client) that can send messages to another (server) using the server's published interface (which is written in some interface definition language, e.g., CORBA IDL). To do this, the support libraries must include XDR convertors (also called “marshallers”) that

take the arguments from the client and turn them into a linear, machine-independent format, and then convert them back for delivery to the server function. The RPC/RMI interface (part of a CORBA ORB's "object adaptor") manages how the remote procedure call is passed over the network, and how the client maps from the network format of the call to an actual program function address.

Traditional socket-based distributed programming tools provide the first two features (XDR and RPC/RMI), but generally lack the latter two (events and QoS). The basic CORBA specification standardizes XDR, RMI, and "event channels" but lacks any notion of QoS.

CORBA also introduces overhead that may make it too inefficient for many real-time, high bandwidth, low latency, bounded latency jitter applications. The CORBA RMI and event mechanisms also have very different performance characteristics and scalability considerations. In applications where deterministic response time (bounded latency) is a requirement, it may be difficult to characterize the suitability of CORBA RMI or event channels.

The original CORBA specification assumed nothing about the run-time performance of ORBs and applications built on top of them; it assumed that the ORB and the host operating system would provide "best-effort" performance, and that applications would not require strict QoS enforcement or even predictable and scalable run-time behavior. As we shall see below, these issues are the focus of much of the recent work on CORBA.

Recent Extensions to the OMG CORBA Specification

The CORBA standard has been revised several times since its inception in the early-1990s. Some of these changes have impacted the core of the ORB structure (e.g., the use of the IIOP protocol, the CORBA real-time specification, or the minimal-CORBA specification), while others have added higher-level services that can be layered on top of an ORB's core functions (e.g., the event service or A/V streaming service). We will discuss the two most relevant new components—real-time CORBA and the CORBA A/V streaming service—next.

The Real-Time CORBA Specification

The real-time CORBA specification (OMG 2001c) describes the interaction between an ORB and the operating system on which it is hosted (rather than focussing on new CORBA IDL interfaces or services). It concentrates on features that can support predictable end-to-end performance for fixed-priority applications. Although a version of R-T CORBA was included in the CORBA 2.4 specification (October, 2000), the newest version incorporates features from the CORBA 3.0 standard (work in progress), specifically the GIOP/IIOP version 1.1 protocol and the QoS model from the CORBA Messaging specification (OMG 2001a). The standard states, "A Real-Time CORBA system will include the following four major components, each of which must be designed and implemented in such a way as to support end-to-end predictability, if end-to-end predictability is to be achieved in the system as a whole:

1. the scheduling mechanisms in the OS
2. the Real-Time ORB
3. the communication transport

4. the application(s)

Real-Time ORBs conformant to this specification are still reliant on the characteristics of the underlying operating system and on the application if the overall system is to exhibit end-to-end predictability.” (OMG 2001c, p. 24-6)

This implies that a full-fledged R-T CORBA infrastructure assumes a real-time operating system and ORB access to the network protocol stacks, as well as requiring a separate global scheduler.

The four central issues in R-T CORBA (according to Schmidt and Kuhns, 2000, p. 2) are:

1. **Communication infrastructure resource management:** An RT-CORBA endsystem must leverage policies and mechanisms in the underlying communication infrastructure that support resource guarantees. This support can range from (1) managing the choice of the connection used for a particular invocation to (2) exploiting advanced QoS features, such as controlling the ATM virtual circuit cell pacing rate.
2. **OS scheduling mechanisms:** ORBs exploit OS mechanisms to schedule application-level activities end-to-end. Since the RT-CORBA 1.0 specification targets fixed-priority real-time systems, these mechanisms correspond to managing OS thread scheduling priorities. The RT-CORBA specification focuses on operating systems that allow applications to specify scheduling priorities and policies. For example, the real-time extensions in IEEE POSIX 1003.1c define a static priority FIFO scheduling policy that meets this requirement.
3. **Real-Time ORB endsystem:** ORBs are responsible for communicating requests between clients and servers transparently. A real-time ORB endsystem must provide standard interfaces that allow applications to specify their resource requirements to the ORB. The policy framework defined by the OMG Messaging specification allows applications to configure ORB endsystem resources, such as thread priorities, buffers for message queueing, transport-level connections, and network signaling, in order to control ORB behavior.
4. **Real-time services and applications:** Having a real-time ORB manage endsystem and communication resources only provides a partial solution. Real-time CORBA ORBs must also preserve efficient, scalable, and predictable behavior end-to-end for higher-level services and application components. For example, a global scheduling service can be used to manage and schedule distributed resources. Such a scheduling service can interact with an ORB to provide mechanisms that support the specification and enforcement of end-to-end operation timing behavior. Application developers can then structure their programs to exploit the features exported by the real-time ORB and its associated higher-level services.

Discussion

We see several issues that prohibit us from using RT-CORBA for HPDM applications at present. First, Schmidt and Kuhns (2001) mention that, “an important class of mission-critical applications cannot meet their QoS requirements under dynamic load conditions just using the features standardized in the RT-CORBA 1.0 specification. Moreover, it is very hard for

complex mission-critical application developers to determine the priorities of various operations *a priori* without significantly underutilizing various resources, such as the CPU. To address these issues, therefore, the OMG is standardizing dynamic scheduling techniques, such as deadline-based or value-based scheduling." Several efforts are underway to extend RT-CORBA and to provide these "missing pieces," however, and we will follow these developments with great interest.

The second issue pertains to the reusability, portability, and flexibility of HPDM application components. The specification states, "By providing the developer with handles on managing resources and on predictability, Real-Time CORBA sacrifices some of the general purpose nature of CORBA to support the development of Real-Time systems" (OMG 2001c, p. 24-2). For HPDM, we desire a simple, malleable, and portable object model that nevertheless scales well to multi-server "soft" real-time applications. For these purposes, RT-CORBA may be characterized as "overkill."

Lastly, we need to support HPDM applications on a range of hardware platforms using off-the-shelf (not necessarily real-time-tuned) ORBs and operating systems. According to one report (Levine, Flores-Gaitan, and Schmidt, 1999, p. 8) that evaluates OS-level support for RT-CORBA, "Our preliminary results indicate that a real-time ORB like TAO, run on a real-time OS like LynxOS or VxWorks, can provide a very predictable and efficient ORB endsystem platform for real-time applications." Their benchmarks on non-real-time operating systems such as Linux, Solaris, and MS-WindowsNT are disappointing, since the RT-CORBA specification assumes that the RT-ORB has access to low-level operating system facilities that are missing on non-real-time-centric systems.

Advantages of Real-Time CORBA

- Avoids several common operating system problems (e.g., excessive buffer copies, protocol stack overhead, priority inversion)
- Provides for resource (thread, memory, socket) management

Disadvantages of Real-Time CORBA

- Requires special ORBs and operating systems (or at least OS patches and drivers)
- Intended for (and only efficient on) real-time operating systems

The CORBA A/V Streaming Specification

In 1997, the OMG released a draft specification of a new service called the Audio/Video streaming service (CORBA A/V, OMG 2000). This has been revised twice since then, the most recent being in January, 2000. It is expected that CORBA A/V will be included in the future CORBA 3.0 specification. This effort came out of the OMG special interest group for the telecommunications industry. The CORBA A/V specification describes the structure of applications that want to use CORBA for some of their data interchange, and also to pass streaming media content data at higher rates and using different protocols than CORBA allows. These signals are "out-of-band" from the perspective of CORBA, but the streams are managed by CORBA objects. The specification (OMG 2000, p. 2-2) states,

“The Control and Management of Audio/Video Streams specification addresses the following issues: Topologies for streams, multiple flows, stream description and typing, stream interface identification and reference, stream set-up and release, stream modification, stream termination, multiple protocols, Quality of Service (QoS), flow synchronization, interoperability, and security. This specification addresses the first 11 issues and provides hooks for solutions to the last issue (security).“

The standard defines an object model for multimedia devices, streams, and flows (which is reminiscent of ATM networking technology). The simplest model for an application uses two virtual devices (VDevs), two stream end-points, and a stream control object, as illustrated in the Figure below (OMG 2000, p. 2-4).

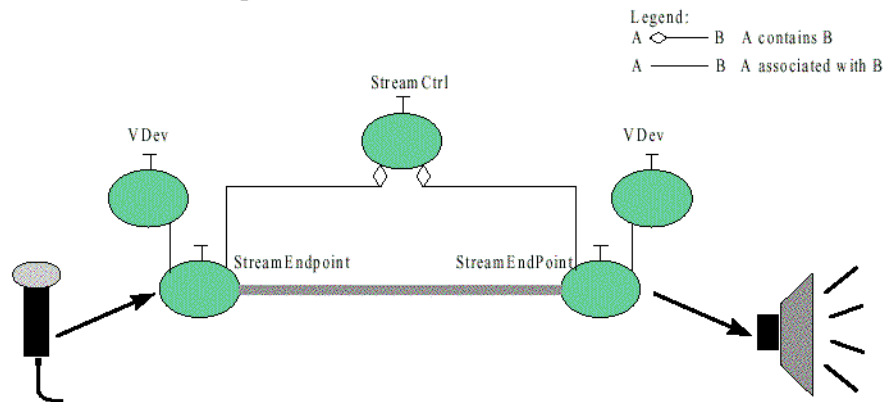


Figure: CORBA-A/V Stream and Device Connections

This is, unfortunately, drastically oversimplified. The canonical diagram of the full CORBA-A/V architecture is shown in the Figure below (taken from page 2-2 of the specification); it illustrates the use of an out-of-band media data flow (the thick line at the bottom of the Figure), which is controlled by CORBA objects representing the stream and flow endpoints (one stream can contain several flows). There are control and management objects used for the streams and for the top-level client-server connection.

The detailed architecture (see [Mungee, Surendran, Krishnamurthy, and Schmidt, 2000] for a good overview) shows that each stream endpoint is decomposed into three objects: a stream interface control object that exports a public IDL interface; a data source or sink (which is assumed to be bound to a virtual device), and a stream adaptor that is responsible for sending and receiving data frames over a network. The various stream and media control objects are responsible for set-up and control of streams and flows. The flows themselves are interpreted using the Simple Flow Protocol (SFP), which is itself defined using the flow specification syntax.

The IDL specification of these objects fills eleven pages in the specification. It details the functionality of the various flavors of virtual devices, stream and flow endpoints, and media and stream control objects, as well as the media streaming framework that supports these, and the application-level and network-level QoS models used in CORBA-A/V. The specification is layered into “light” and “full” profiles, with the light profile being a proper subset of the full profile. The light version does not include flow connections, flow endpoints, the flow protocol, or flow devices (FDevs), and is also missing the media control interface.

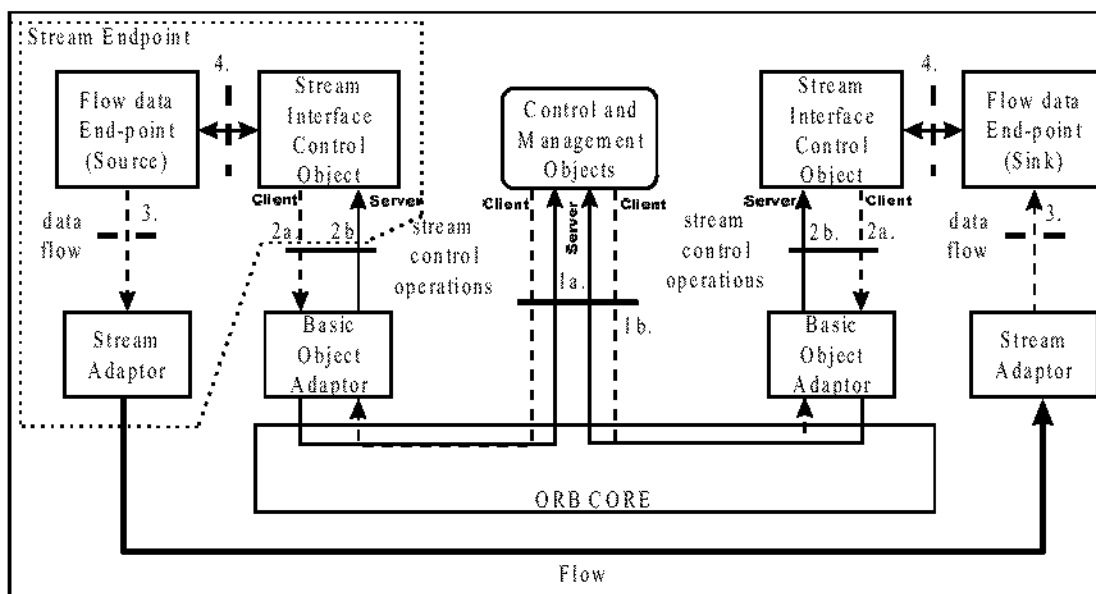


Figure: CORBA-A/V Architecture

CORBA-A/V includes two sets of QoS parameters, one mandatory and one optional. Both of these are network-level parameters such as delay, bandwidth, jitter, and cost. The second parameter set includes such properties as policy flags for how to handle duplicate, damaged, out-of-order, or lost packets, and supports the description of min/avg/max data packet sizes.

Discussion

Thankfully, the specification (and related references) includes numerous pseudo-code examples of stream setup and management, and extensive message sequence charts for the various protocols and operations. The one implementation of CORBA-A/V to which we had access (ACE/TAO), also includes several examples and benchmarks. Sadly, to our knowledge only the TAO ORB currently supports the CORBA A/V service. Our team invested quite a good deal of time of time in using TAO ORB and making their CORBA A/V implementation run, with only very limited success. The ORB has a serious (and well-known) space leak that kept our benchmarks from running, and we were unable (after several attempts) to get the CORBA A/V examples and benchmarks to compile and run on Linux, Solaris, or MS-WindowsNT. Without repeating the colorful language that several of our team members used to describe the experience, suffice it to say that trying to use CORBA A/V on TAO is a very complex task.

As with the RT-CORBA extensions discussed above, the CORBA A/V streaming standard simply seemed to be “overkill” for our purposes. In addition, there is no support (yet) for using CORBA A/V from any language aside from C++, or on any platforms not supported by TAO. If, at some future point, off-the-shelf ORBs were to support reliable and flexible implementations of CORBA A/V, we would certainly revisit the decision, but for the moment, it looks too expensive (in terms of development effort) for HPDM.

Advantages of CORBA A/V

- Model of CORBA objects managing out-of-band streams
- Abstract models of devices, flows, and streams
- Protocol independence/pluggability

Disadvantages of CORBA A/V

- Not supported by most ORBs (to date)
- Extremely “heavy-weight” (i.e., complex)
- Complex mapping between application-level and network-level QoS parameters
- Very few (any?) real-world applications (to date)
- HPDM applications would require both the full profile specification and the extended (optional) QoS parameter set

Performance of DOC Frameworks

There is a rich literature comparing various aspects of CORBA ORB implementations (see e.g., Amar and Bensoussan, 2000, and MLC Systeme, 1999) and we have performed our own interoperability and performance tests as part of the 1996-7 IDOT project (IDOT, 1997) and, more recently, within the ATON project. It has been our experience that some off-the-shelf ORBs are indeed suitable for HPDM applications; based on this (and on the independent benchmarks we cited above), we have grouped the ORBs that we evaluated into three groups as follows:

Slow:	Orbix, BEA M3, ORBacus, SmalltalkBroker
Fast:	VisiBroker, MICO, OmniORB2
Real-time:	TAO ORB, Vertel e*ORB

The two figures below illustrate the relative performance of modern ORBs; the left-hand graph (from Amar and Bensoussan, 2000) displays the number of calls per second made to interfaces of varying complexity using several ORBs (larger is better). The right hand graph (from AT&T, 1999) compares the round-trip times for calls within a machine or across machines (smaller is better). In both cases, there is a factor of two or more difference between the best off-the-shelf ORBs and the “default” performance of non-optimized ORBs. One can also see (from the left-hand graph, and the report from which it came), that well-built off-the-shelf ORBs can rival (or even equal) the raw speed of the best (and most complicated) research platforms (e.g., TAO ORB).

The data in the graphs below agrees in general with our own benchmarks in convincing us that we can use standard CORBA techniques for the current round of HPDM applications; off-the-shelf ORBs can support 2 msec or less method invocation latency and several MB/sec of packet throughput.

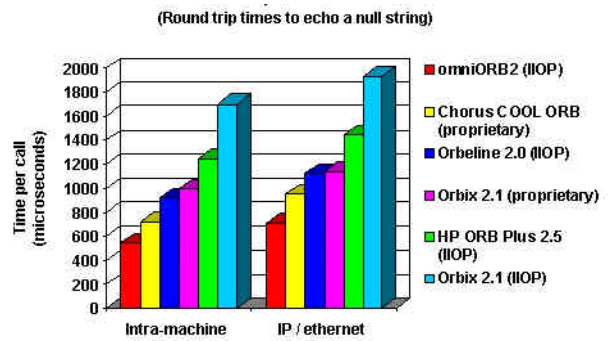
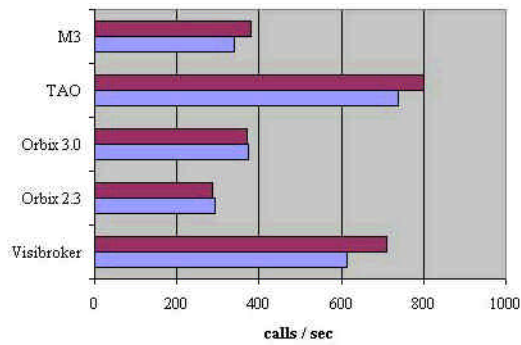


Figure: Third-party ORB Performance Graphs

Options for Deploying HPDM Applications

The initial estimations we made in the HPDM/DPE design report motivated us to explore both the Real-Time CORBA and COREBA A/V standards in detail. We also built a prototype of the DPE management tool, as reported in ATON report 2001.06.5.

Based on our experience running complex applications that consist of several “soft real-time” CORBA services (e.g., for head- and hand-tracking, robot and camera control, and traffic simulation interfaces), we are reassured that we can satisfy our current (and even medium-term future) needs well using off-the-shelf components such as the MICO and SmalltalkBroker ORBs, 100BaseT networks with TCP/IP, and general-purpose operating systems. As our needs grow in the future, e.g., supporting more users, more channels of video, and more complex spatial sound, we have to remain willing to re-evaluate this decision, however.

Conclusions

Within the ATON project, we have implemented a complex multi-server real-time distributed application. We chose to do this using off-the-shelf components as much as possible, and to base our work on stable, widely supported aspects of the CORBA standard. If real-time CORBA research were our mission, we would probably have opted for a more special-purpose solution, but for the more limited requirements of our applications, and given our focus on system-building, we feel that this was the right decision for the ATON Project.

References

Amar V., and P. Bensoussan. 2000, *CORBA ORB Benchmarks*. See <http://beust.com/virginie/Benchmarks/>

AT&T. 1999. *OmniORB2 Performance Report*. see <http://www.uk.research.att.com/omniORB/omniORBPerformance.html>.

DRIVE (Distributed Real-Time Interactive Virtual Environments) Project Documentation. 1997. See <http://www.create.ucsb.edu/drive/phase2/report.html>.

Feng, W., U. Syyid, and J.-S. Liu. 1997. “Providing for an Open, Real-Time CORBA.” in *Proceedings of the IEEE Workshop on Middleware for Real-Time Systems and Services*.

IDOT (Impact of Distributed Object Technology Using CORBA) Project Documentation. 1997. See <http://www.create.ucsb.edu/idot/report.html>.

Kalogeraki, V., P. Melliar-Smith, and L. Moser. 1997. "Soft Real-Time Resource Management in CORBA Distributed Systems," in *Proceedings of the IEEE Workshop on Middleware for Real-Time Systems and Services*.

Krishnamurthy, Y. et al. 2001. "Integration of QoS-enabled Distributed Object Computing Middleware for Developing Next-generation Distributed Applications." *Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001)*

Levine, D, S. Flores-Gaitan, and D. C. Schmidt. 1999. "Measuring OS Support for Real-time CORBA ORBs." *Proc. of the 4th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS'99)*, Santa Barbara, California, January, 1999.

MLC Systeme. 1999. CORBA Comparison Project Final Report. see <http://www.mlc.de>.

Moser, L. E., and P. M. Melliar-Smith. *The Eternal System*. See <http://alpha.ece.ucsb.edu/eternal/Eternal.html>

Mungee, S., N. Surendran, Y. Krishnamurthy, and D. C. Schmidt. 2000. "The Design and Performance of a CORBA Audio/Video Streaming Service." in Mahbubur Syed, ed. *Design and Management of Multimedia Information Systems: Opportunities and Challenges*, Idea Group Publishing, Hershey, PA.

O'Ryan, C., et al. 2000. "Evaluating Policies and Mechanisms for Supporting Embedded, Real-Time Applications with CORBA 3.0." *Proceedings to the Sixth IEEE Real-Time Technology and Applications Symposium (RTAS'00)*.

OMG 2000: CORBA A/V Streaming Specification, see <http://www.omg.org>.

OMG 2001a: CORBA Messaging Specification, see <http://www.omg.org>.

OMG 2001b: Minimum CORBA Specification, see <http://www.omg.org>.

OMG 2001c: Realtime CORBA Specification, see <http://www.omg.org>.

Schmidt, D. C., and F. Kuhns. 2000. "An Overview of the Real-time CORBA Specification." *IEEE Computer special issue on Object-Oriented Real-time Distributed Computing*, June 2000.

Zinky, J. A., D. E. Bakken, and R. Schantz. 1997. "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems*, 3(1).