Real Time Platform Middleware for Transparent Prototyping of Haptic Applications

George Pava & Karon E. MacLean
Dept. of Computer Science
University of British Columbia
Vancouver, B.C. Canada
E-mail: {pava, maclean}@cs.ubc.ca

Abstract

In this paper we present The RealTime Platform Middleware (RTPM), an n architecture forsupporting the development of complex, prototyping distributed realtime collaborative Haptic Applications.multimodal I/O projects.

Complex HapticMultimodal aApplications often require a distributed implementation across multiple computers in a network to fulfill meet the timedisparate temporal and platform constraint for haptic rendering and visualization processes.needs. RTPM

The Real Time Platform Middleware (RTPM) is a Distributed Object Computing (DOC) middleware that provides an extendable, device-independent, and network-transparent interface to a collection set of user I/O devices which. eases application integration across different operating systems.

RTPM consists of a framework that usesbased on Common Object Request Broker Architecture (CORBA) and a custom Virtual Device abstraction. The Virtual Devicethat is the component that is the front end for the real device and exports its real devices' functionality to the userClient applicationsprocesses. The RTPM framework It offers two mechanisms (Client/Server and Consumer/Supplier) mechanism for communication between between user components processesin a complex Haptic Application.

This paper describes the architecture's objectives and implementation, provides examples of its use and analyzes its performance in some typical haptic application configurations.

RTPM reduces the programming effort necessary to interface with I/O devices, and integrate with existing systems and create scalable Haptic Applications.

Keywords:
MMultimodal applications, dDistributed sSystems, architecture, rReal-tTime operating sSystem, Architecture, CORBA, …

1. INTRODUCTION

The creation of a high-performance haptic application poses special demands of software architectures and operating systems. Haptics researchers developing new prototypes often find themselves pushing the limits of available operating systems and of fast, deterministic network connectivity. The difficulties escalate for applications which integrate multiple I/O modalities: these generally require a variety of refresh rates and consequently multiple threads or processes (if not multiple platforms), which must synchronize timing and share data, often among computers.

The needs for complex haptic applications – rarely *all* well met by any one operating system – include:

- A relatively high and consistent haptic refresh rate
- Synchronization of a fast haptic process with slower refresh cycles for other I/O modalities – usually graphics or audio, but also reflecting diverse needs such as motion capture, eye tracking and biometrics.
- Management of data shared among processes
- Interprocess communication, e.g. event notification
- Reliable inter-CPU communication
- Easy scalability when the application's computational needs exceed that of a single CPU
- Integration of legacy systems and specialized I/O hardware, particularly when these are implemented on different operating systems.

Traditionally, research developers of haptic applications have produced custom, dedicated solutions, often with an extensive investment of effort and yet a result that is monolithic rather than modular, limited in scope, difficult to extend and tied to a single platform. While some specialized realtime operating systems are better suited to certain haptic application needs, their very strength – the low-level system control which is the key to, and the price of realtime performance – as well as their relative obscurity impede the integration of other application aspects for which better support may exist in mainstream operating systems.

Proceedings of the 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'04)
0-7695-2112-6/04 $20.00 © 2004 **IEEE**

1.1 Goals of This Project

Our group needed to create a modular, easily extensible but high-performance platform that would support quick prototyping of setups for a wide variety of "fast" multimodal psychophysics experiments, and for applications that exploit their results. Given the wide variety of computer platforms used by our collaborators and the range of I/O devices used, there was clearly no single platform that would satisfy our needs.

We therefore explored the general concept of a fast local area network (LAN) combined with a custom middleware layer that allows modules resident on multiple computers (of arbitrary operating system) to communicate with one another *as if they are on the same computer*. That is, we required that:

- A given pair of communicating modules could be installed on the same computer or on two different computers with no alteration in their structure, communication parameters or data management; and
- The latency and throughput of inter-module communication would meet specified levels (suitable for haptic refresh rates, generally the most demanding) when separated by a LAN.

These specifications together ensure superior flexibility as to operating system and extensibility, while providing support for data management, interprocess communication and other typical realtime application needs. Thus, processes with demanding real time needs may be located on a CPU running a specialized real time operating system (RTOS), yet be easily integrated with processes already implemented on more common systems. This solution is appropriate for the research developer who is prototyping multimodal systems: the number of units is low, while time and flexibility are often of greater value than the setup's component cost.

1.2 Our Approach

We have developed the "**Real Time Platform Middleware**" (**RTPM**), which augments a realtime implementation of an existing middleware protocol - CORBA, or "Common Object Request Broker Architecture" [10] – with a family of custom elements that tailor it to high-performance, multimodal application needs. The CORBA specification allows applications to interoperate relatively transparently over networks, and because there are implementations (ORBs, [8]) for many operating systems and languages, it is often said to facilitate platform independence.

Our project uses a CORBA implementation which began as the "Adaptive Communication Environment" (ACE) framework [13], providing realtime wrappers for network applications; was extended in the mid-90's with CORBA modules to become "**T**he **A**CE **O**RB" (**TAO**) [15]; and formed the prototype for CORBA's recent realtime extension (RT-CORBA, [6]). Its relevant aspects are described in Section 2.4.

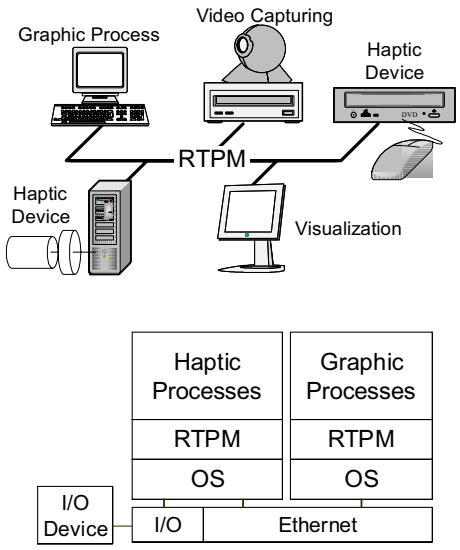


Figure1: Realtime Platform basic architecture

An example of RTPM's basic system architecture is illustrated in Figure 1. An arbitrary collection of soft- and hard-realtime processes can run on the same or different "nodes" connected over a Local Area Network. These nodes may run any supported operating system – e.g. Linux, Windows, MAC-OSX. RTPM manages inter-process communication among nodes, such that running two processes on the same vs. different nodes makes little difference either code- or performance-wise. Finally, RTPM provides data services from I/O devices connected to any node: data from/to these devices is available to processes on any other node.

RTPM thus creates a "transparent" approach in the following senses:

- **Location transparency**: different modules (e.g. data acquisition, model update, rendering) can run either on the same machine or distributed in a LAN.
- **I/O transparency**: an application can access similar-function I/O devices with radically different native interfaces through a common interface.
- **Language transparency**: applications written in different languages (e.g. Java or C++) can access I/O devices through the same interface.
- **Scaling transparency**: either I/O devices or computational modules can be added with minimal effort. Integrating a new device means writing a standardized interface module; application modules need not change when relocated to a different machine.

1.3 Related Work

The temporal constraints imposed by the haptic servo's refresh rate and its synchronization with softer

realtime processes has led to distribution solutions that vary from single- or multithreaded, uni-process architectures, to the use of dedicated interprocess communication mechanisms available on specialized platforms.

For example, Vahora et al. [16] solved the realtime issues for virtual reality haptic applications by using named pipes available on Windows NT for interprocess communication. Other distributed haptic applications have used proprietary communication infrastructure to connect a haptic rendering with a graphic process, as described by [2, 5, 11]. MacLean et al [4] described a custom architecture appropriate for simple haptic processes requiring tight integration with other distributed I/O processes.

While often providing adequate performance, these approaches were hard to create; and are hard to scale in size and I/O capability and to port to other platforms.

While CORBA has been a potential solution to some of these needs, early versions were unsuitable for distributed realtime applications. However, the recent (2002) introduction of the RT-CORBA specification [6] and the TAO implementation of the Realtime Event Service [7, 12], along with ever-faster computation, has in just the last year given us an architecture model and system components that make feasible the prototyping toolkit for distributed multimodal systems described here.

1.4 Remainder of This Paper

In Section 2 we describe the main components of the RTPM architecture and explain how it addresses key haptic realtime design issues. In Sections 3-4 we present two example implementations using RTPM and offer guidelines for using RTPM; Section 5 offers performance results for different platforms and configurations. We end with our conclusions and future work.

2. ARCHITECTURE

Our combined goals of software reuse and the kind of process and node distribution required for I/O and processing by most haptic and multimodal applications has led to a three-layer approach (Figure 2). The top layer consists of **Applications**, and the bottom of **Operating System and Hardware**. **RTPM**, the middle layer, provides an environment for easy implementation and integration of the other layers, and a run-time software bus for communication between them.

2.1 Application Layer

The top layer contains the specific processes employed by the application. Some of these modules carry out application-specific tasks, but others may be reusable. E.g, a haptic application might include modules that get device position, detect collisions, send force commands and log data; a graphic component application would make use of an analogous process set that are in the same

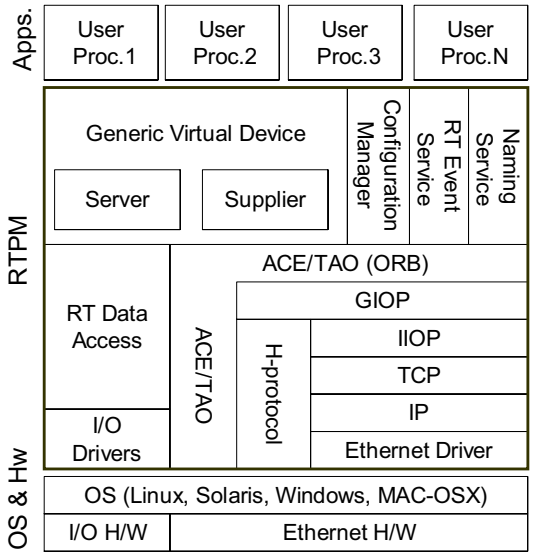


Figure 2: RTPM architecture overview

layer, but might be located on a different node. There are reusable prototypes for all of these common tasks. Two specific application examples are described in Section 3.

2.2 RTPM Layer

Our Realtime Platform Middleware (RTPM) is an integrative framework that supports two complementary models of object-oriented communication and data sharing among the distinct sets of objects contained in the top and bottom layers: Client–Server or Consumer–Supplier relationships. From the perspective of the programmer, there should thus be no difference between invoking an RTPM service, or calling into a static library – an illustration of RTPM's location transparency.

The main modules of the RTPM are as follows. Of these, the first three were created in the course of this project, and the fourth (H-Protocol) is a potential avenue for RTPM's future expansion.

- **Generic Virtual Device (GVD):** provides a core set of commonly used interfaces for I/O devices used in multimodal applications. GVD represents an abstract I/O device that is referenced by the application layer in the same manner as a static library. Operations invoked on the GVD by an application-layer call are redirected to the real device that actually carries out the command, within and across nodes.
- **Configuration Manager (CM):** Some device functions – e.g. I/O device operations such as serial communication parameters, analog calibration, etc – must be performed before the device is addressed or even during its operation. CM provides a set of interfaces for configuration/calibration, either programmatically or through a client GUI application.
- **RT Data Access Layer (DA) and I/O Drivers:** The I/O devices used in haptic/multimodal applications (e.g.

different haptic displays) use a disparate set of device drivers designed for different platforms and typically with nonstandard API's. However, most of these low-level drivers perform essentially the same functions: namely opening, closing and configuring the device, and reading/writing its data. DA provides a unified and extendable set of interfaces to this common functionality. New devices are integrated by creating an interface module from an existing template.

- **H-Protocol:** If CORBA communication timings must be improved in the future, H-protocol provides a path to architect our own alternative transport protocol using TAO's open pluggable protocol framework [1], replacing the default Internet Inter-ORB protocol (IIOP) used for inter-orb communication. Figure 2. depicts the standard protocol stack used by TAO:
  - **GIOP**: General Inter-ORB Protocol (for messaging)
  - **IIOP**: maps GIOP on to TCP/IP
  - **TCP/IP**: transport and internet protocol
- **TAO ORB and RealTime CORBA extension:** The TAO ORB delivers client requests to the server and returns responses to clients. TAO's realtime ORB core uses a multithreaded, preemptive, priority-based connection and concurrency architecture, for efficient and predictable client server communication [6, 15].

TAO's implementation provides the following CORBA services:

- **RealTime Event Service (RTES):** defines an event data delivery model that decouples communication between suppliers and consumers of events [7, 12].
- **Naming Service (NS):** registers RTPM server objects with names, and comprising the principle mechanism by which clients locate objects they intend to use [9].

2.3 Operating System and Hardware Layer

The bottom layer consists of the various nodes' operating systems, network and non-graphics I/O device hardware. Graphics I/O is still handled through platform-specific non-RTPM routes to the OS layer.

Because of the mediation provided by RTPM, most application-layer processes using RTPM require only re-compilation to run on RT-CORBA-supported platforms, which at present are Linux (including some realtime variants), Windows, and MAC-OSX. When thus redistributed, a haptic device can connect to a PC-Windows 2000 and communicate with a graphics process executed from an Apple-MAC-OSX.

Any platform-dependent calls within the application layer – e.g. graphics methods available in Linux but not Windows – must obviously be ported. This can generally be avoided with the haptics components, since RTPM services its I/O and synchronization.

2.4 CORBA Background

Critical to understanding the architecture described here is a basic knowledge of CORBA and its realtime extension, RT-CORBA [14]. The latter defines several mechanisms to provide tight control over quality-of-service characteristics such as jitter and latency.

Threads: RT-CORBA uses threads as a schedulable entity. Generally, a thread represents a sequence of control flow within a single node. Threads are part of an activity. Activities are "scheduled" by coordinating their constituent threads.

Threadpools: RT-CORBA defines the Threadpool abstraction to manage server-side execution threads.

Lanes: A threadpool may be created with a single default priority for all its threads, but RT-CORBA also supports multiple thread priorities within a single threadpool. Threads within a threadpool with the same priority are grouped into "lanes". Each threadpool lane has it own configuration for static and dynamic threads.

Priority scheme: RT-CORBA defines a universal, platform independent priority scheme called *RealTime CORBA Priority*. This is introduced to overcome the heterogeneity of different operating system priority schemes, and allows RT-CORBA client applications to make prioritized CORBA invocations in a consistent fashion between nodes with different priority schemes.

GVD uses RT-CORBA's "Client Propagated Priority" model: a client (e.g. a haptic process) sets an invocation's RT-CORBA priority which is propagated to the server ORB, which in turn propagates it into its own native priority scheme. Requests from non-RT-CORBA ORBs (i.e. ORB's that do not propagate an RT-CORBA priority with the invocation) are handled at a priority specified by the server – in our case, GVD.

Event Service: CORBA's Event Service provides support for decoupled communication between objects, allowing suppliers to send messages to one or more consumers with a single call. Suppliers need not be aware of any of the consumers of its messages; the Event Service mediates this communication and also shields suppliers from exceptions resulting from a consumer object being unreachable or poorly behaved.

TAO's realtime version of the CORBA Event Service (RTES) [12] includes these features:

- **Event filtering -** consumer processes may register for event delivery based on event type or supplier id (e.g. events from a particular I/O device). The event channel filters events based on these registrations, to ensure efficient event delivery.
- **Event correlation -** consumers may register for event delivery based on conjunctive or disjunctive sets of events. Conjunctive registrations cause the event channel to notify the consumer when all events in the set have arrived. Disjunctive registrations cause the

event channel to notify the consumer when any event in the set has arrived.

- **Periodic event processing -** consumers may register for suppliers, based on timed events. RTES then generates "Timeout events" that can be integrated into the event filtering scheme.

2.5 Alternative Application Models

Applications making use of RTPM's GVD can be based on two different models depending on their needs.

Client-Server Model: synchronous communication for client applications with hard realtime requirements. A client – for example, a haptic, graphic or audio process – makes requests to the server (GVD) to execute certain operations (e.g. get position, set force, etc). The client then blocks and waits for the reply; meanwhile, other clients can simultaneously access the same GVD. Priority of GVD access is mediated through RT-CORBA's Client Propagated Priority Model, described above.

Consumer-Supplier Model: asynchronous communication for consumer applications with softer realtime requirements. For processes that require neither high frequency updates nor tight integration with data-generating objects (e.g. graphic display and data logging), this model can be more efficient through its support of broader data distribution.

GVD's Consumer-Supplier model is implemented with the RT-CORBA Event Service, described above, and utilizes RTES event filtering, event correlation and periodic event processing.

2.6 Realtime Issues

Failure to meet timing constraints directly impacts haptic rendering performance. With contemporary computers, meeting the requirements for average update rates is not usually the problem except for very complex models. These are generally accepted to be a haptic refresh of 0.5-1kHz, and a graphic refresh of 30-70 Hz.

Instead, our most important real time characteristic is often predictability – i.e., the system must *always* satisfy specified timing boundaries.

An OS attempts to maintain appropriate use of CPU resources through its scheduling strategy, scheduling threads by criteria such as priority and resource usage. Any scheduling strategy must trade off fairness for increased predictability and control by some threads. TAO allows applications to specify their own scheduling priorities and policies.

RTPM addresses predictability through activity prioritization. It is configured to use a FIFO (first-in-first-out) scheduling policy. That is, the highest priority thread runs until it either voluntarily yields control, or is preempted by a realtime thread with a higher priority.

The application processes and their invocations on GVD can be configured to run at different priorities, so that the hard realtime system components will behave deterministically and critical tasks will receive high priority in their execution. RT-CORBA defines 32768 priority levels, which are mapped transparently onto the native (OS- specific) priorities.

It is important to understand that RT-CORBA relies on the underlying operating system for scheduling its tasks, and therefore the hard realtime characteristics can only be achieved if the OS can deliver this. Threads must take turns running on the CPU so that one thread doesn't prevent other threads from performing work. One of the OS scheduler's jobs is to assign units of CPU time (*quantums)* to threads. A quantum is short in duration, but threads receive quantums so frequently that the system appears to run smoothly, even when many threads are performing work. One difference between the OSs we tested (Linux, Timesys and Windows 2000) is the length of a user thread's quantum. On most x86 systems running Windows 2000 as well as standard Linux, a quantum is 10 milliseconds (ms); only the realtime Linux, Timesys, has a very short quantum of 20 microseconds (μsec).

3. EXAMPLE APPLICATIONS

3.1 Haptic Servo Application

Our first example is a prototypical haptic control scenario, where a single haptic device is managed, controlled and visualized from two three other different devices managed and visualized by one node is controlled from a second nodes via RTPM. . The control data for this single-degree-freedom device consists of an encoder reading and a force commanIn the configurationd.

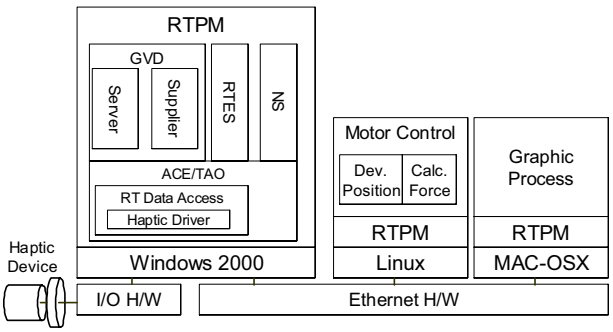


Figure 3: Haptic servo application

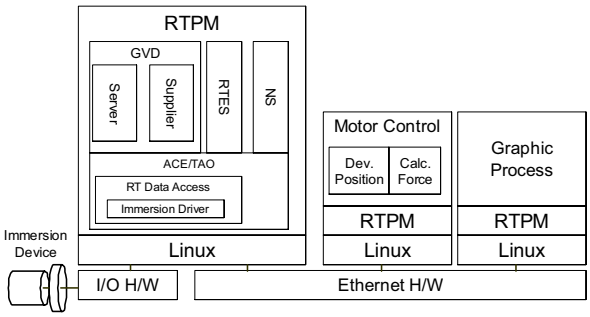


Figure 1: Haptic servo application

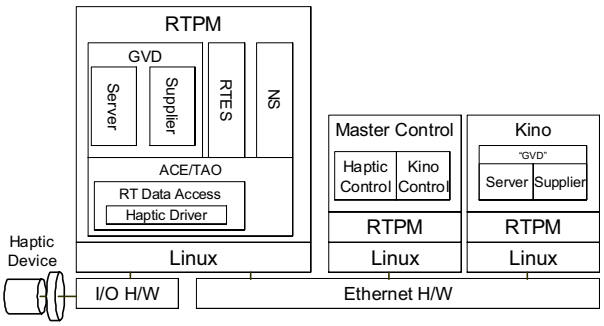


Figure 44: Haptic control of media application

In the configuration of Figure 3, the haptic device is connected to node is a Windows 2000 machine, the motor control code is executed on a Linux CPU running Redhat Linux., and the visualization – motor positiona graphics process, is implemented on an Apple laptop running MAC-OSX. T.his degree of network distribution may be desirable if the task executing on one node is computationally intensive, or utilizes special platform strengths This division is artificially made to demonstrate how we can divide a Haptic application into components that can be distributed across the network. These components can run on different machines, and on different Operating Systems (Linux, Windows, MAC_OSX)..

The motor control code (Linux node) is The Motor Control is a CClient for the GVD, requesting the current motor encoder position and sending the force feedback command. The haptic device itself is opened as a GVD Sserver object is and registered in the CORBA naming service (NS), soand thereafter
theany GVD clients can getrequest its referencesthe object's data across the network. The GVD also acts as a Supplier, sending every new position to the RTES, which resends it to the subscribed consumer processes. RTES' consumers can configure it for event filtering, event correlation or time-out events. In this example, the graphics process is a GVD Consumer, The Graphic Process is a Consumers for the GVD, receiving periodic event events that represent updates of the motor encoder position. The Supplier component of the GVD sends every new position of the motor to the RTES, which will resend this data to the consumer Graphic Process. RTES can be configured
by its consumers (in our case the Graphic Process) for event filtering, event correlation or time-out events.

The Data Access Layer in this first RTPM example application consists primarily ofis merely a wrapper for the haptic display's original Immersion APIWindows driver.

3.2 Haptic Control of Media Application

We needed to interface a haptic interface to an existing Linux open-source video editing application (Kino [3]), and wanted to minimize modification of the third-party source code.

RTPM facilitated a minimally invasive integration of the two original applications, as well as their distribution on two nodes to optimize realtime performance and platform dependencies.The scope of this project is to create a haptic interface for a video editing application. For the video editing software we chose Kino open source video editing. One possible solution was to extend Kino with a Haptic interface by integrating into Kino source code the actual functions for controlling the Media Controller device. Any experimental haptic models would have been hard coded, embedded into Kino application. However, this is not a scalable solution, as any new upgrade on Kino software would have made the reintegration of haptic modules very tedious.
The only modification made to Kino was addition of a In our solution architecture we made use of the RTPM for Kino and Media Controller interface. A modified GVD component has been added to Kino application to provide a both Server and/ Supplier functionality, thus allowing Kino to communicate through GVD (Figure 4).

y; through this components we export the required Kino functions with a minimum intrusion into Kino's architecture.

For the Media Controllerhaptic device interface we used athe same similar setup as described in the previous example:, a GVD that shields hides the details of the actual device.

The Master Controller (MC) application-layer process, which coordinates the interaction between Kino and the haptic display, has two roles:

(a) The Haptic Control subcomponent is a Client for the haptic GVD, performing the same haptic control function as in the previous example.

(a) The Kino Control subcomponent is a Client for Kino's is the new component where we will integrate the haptic models for our experiments and….???KM???…. MC is a Client and a Consumer for both Kino and the GVD. The Haptic Control (HC) subcomponent interacts with GVD for Media Controller position reading and force feedback transmission. Kino Control (KC)

subcomponent interacts with Kino Application for video editing controlGVD server, and via this route can send commands such as (setting video play rate
(b) this is the focus of our haptic model experiments.

The overall performance achieved in this case is perceptually adequate: video motion commands generated by the haptic display are displayed graphically without perceptual delay.

4. USING RTPM

RTPM is designed for ease of integration of client applications. The following describes in more detail how this is done at the Application and OS layer levels.

4.1 Application Layer

The simple client program example in Figure 5 illustrates how to open a haptic device, set the priority of its servo thread, read a position and return a force feedback command.

This program first constructs a GVD client object (vd1) as a wrapper for the remote haptic device. Any invocations on this object will be applied transparently to the real device. It then instantiates a consumer object (consumer1) to receive device state updates.

Once the haptic device has been opened, the vd1 object is used to get the current device position and send a force command in a Client/Server relation. In this example, we run getPositionX() at a higher priority than getPositionY() merely to demonstrate changing the priority level on the server side.

The consumer1 object is for use by other processes that need to access this device's input and/or output asynchronously, i.e. in a Consumer/Supplier relationship – e.g. for graphic or auditory display. Here, we simply print out the haptic device's position information.

```
int main( )
{
  GVDClient          vd1( ) ;          // GVD Client
  GVDConsumer    consumer1( );   // GVD Consumer

  DimX   posX, forceX;
  DimY   posY;

  vd1.openDevice(devA);    //start device "devA"
  while (1)
   {
    vd1.setPriority (3);
    posX = vd1.getPositionX(devA) ;
    vd1.setPriority (1);
    posY = vd1.getPositionY(devA);
    forceX = calculate_force( posX, posY);
    vd1.setForceX(forceX, devA);
    ....
   }
  vd1.closeDevice(devA);  //stop device
  ....
}

// Override the push() operation for GVDConsumer
void GVDConsumer::push(const RtecEventComm::EventSet& events)
{
  ...
  // Loop through the events
  for ( i = 0; i < events.length (); i++)
  {
    if (events[i].data.any_value >>= positionX)
      cout <<"New position = "<< positionX << endl;
  }
}
```

Figure 55: Using RTPM

4.2 OS & Hardware Layer

Connecting an RTPM application to the OS layer and launching it is likewise straightforward. If the new application can make use of existing GVD operations and I/O device interfaces, then we simply start a script that starts first the Naming Service, then the Realtime Event Service and finally the GVD Server / Supplier.

If the application requires new GVD functionality, then GVD's interface definition language (IDL) file must be modified by creating the new operations or modifying existing ones. GVD's two main components, Server and the Supplier, must then be updated to reflect the new or modified parameters or operations.

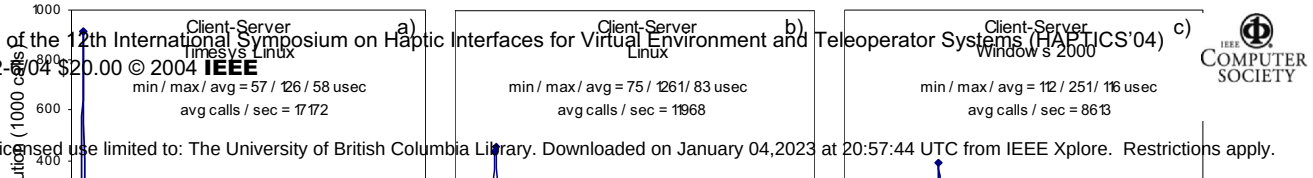
5. PERFORMANCE

In order to verify that RTPM was meeting our performance specifications for typical haptic and multimodal applications, we conducted some benchmark tests based on distributed control of a haptic device through a one- or two-node configuration, comparing several different node platforms. We were particularly concerned with the latency and jitter in communications among the different components.

In these tests, we employed a data structure of six doubles (48 bytes) as our "payload" for transfer between the distributed components in a single refresh cycle. All tests were run on either one or two identical machines: a single-CPU 2.5 GHz Intel Pentium IV system configured with 512 Mb of RAM and a 256Kb cache. The two machines were connected via standard 100BASE-T ethernet on our department local area network, carrying a normal network traffic load.

In a single test run of the Client/Server model, a client thread issues 1000 calls at the fastest possible rate; we then record the round-trip delay experienced by the client from the time it sends a given request to the time it receives the response from the server thread (1000 values). Figures 6-7 present these results as latency distributions with summary statistics. Figure 6 shows the Client/Server results for a single machine running Timesys, Linux or Windows 2000. Inter-process communication is performed via the network loop interface because the client and the server process run on the same machine.

In Figure 7-a), the Client and Server processes are running on different machines; thus comparison with Figure 6 reveals the delay introduced by the network.



Proceedings of the 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'04)
0-7695-2112-6/04 $20.00 © 2004 IEEE

The comparable test for the Consumer-Supplier communication is presented in Figure 7-b), c). In these tests, the Supplier process generates 1000 events (calls) with 1 ms interval between them, and pushes the events to the Event Channel process. EC resends these events to the appropriate Consumer process. The event packet contains the time-stamp plus some overhead (source ID, type, etc.), in total ~30 bytes. The Consumer compares the timestamp of the event with the time at arrival and calculates the transmission delay. Both Consumer and Supplier processes run on the same node (to allow us to compare the time to the necessary accuracy), but the Event Channel process runs on the second node in order to model the delay introduced by the network.

The average single-node latency for Client-Server communication is similar on all platforms, with the best result given by Timesys where 91% of the calls were 58 μsec long. For the non-realtime platforms W2K offers a better distribution but Linux has a better average timing; 99.8% of these results were within the 1KHz specification for haptic processes. The network introduces an extra delay of about 110μsec when Client-Server processes running on Timesys are distributed. Similar values result for other OS combinations, with a worst case of 200μsec.

Dual-node tests for the Consumer-Supplier communication demonstrate this delay as well as a much wider distribution that is introduced by the third process, the Event Channel.

It is important to note that these tests focus on the delay introduced by the CORBA communication layer in our haptic applications, using common operating systems and network setups. There are many other important aspects to determining overall realtime performance; for example, the time quantum required for thread switching plays a much greater role when multiple threads are contending for CPU services – unlike the situation here. The overall round-trip system response time will also be influenced by the delays generated by other RTPM components (DA, CM and I/O drivers) and their synchronization, and by application processes

6. CONCLUSIONS & FUTURE WORK

The timing results from our experiments demonstrate the feasibility of using a distributed object computing architecture for multimodal applications: complex applications can be decomposed and run as separate processes and distributed across a network with an acceptable impact on performance, for a scalable and maintainable system. The Client-Server architecture, gives the best timing results when running on a realtime OS, and is the model of choice for haptic rendering processes that require fast updates. The Consumer-Supplier architecture is more suitable for the decomposition of the slower processes that require an asynchronous type of communication, such us graphic updates and data logging.

We ran most of our tests on Linux and Windows platforms, which are not realtime operating systems. Predictably, our best single-node results are for the one realtime OS tested. However, all provide adequate performance for distributed haptic applications, and system performance can be improved by choosing the Client-Server architecture for the time critical path, creating an isolated network of nodes running realtime operating systems.

We plan to further extend RTPM by adding new components such as CORBA's Audio/Video Service (AVS) and Trading Service (TS). AVS is important for audio/video data exchange, and TS for client queries of existing, active GVD functions. We will be extending the Configuration Manager and the Data Access Layer to accommodate interfaces and control for more haptic devices. Finally, for collaborative haptic applications over the Internet and for some graphic processes implementations, we will add Java language support by integrating the Java (J2EE) ORB with RTPM.

7. REFERENCES

[1] Arulanthu, A. B., O'Ryan, C., Schmidt, D. C., M.Kircher, and Parsons, J., "The design and Performance of a Pluggable Protocols Framework for Real-time Distributed Object Computing Middleware," in Proc. of IFIP/ACM, Middleware 2000, 2000.

[2] Jordan, J., Mortensen, J., Oliveira, M., and Slater, M., "Collaboration in a Mediated Haptic Environment," in Proc. of PRESENCE 2002: the 5th Ann. Int'l Workshop on Presence, University Fernando Pessoa, 2002.

[3] Kino, DV editor for GNU/Linux, 2003. http://kino.schirmacher.de/article/static/2.

[4] MacLean, K. E. a. S., S. S, "An Architecture for Haptic Control of Media," Proc. of the 8th Ann. Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems, ASME / IMECE, Nashville, DSC-5B-3, 1999.

[5] McLaughlin, M. L. et al., "Performance and co-presence in heterogeneous haptic collaboration," in Proc. of the 11th Ann. Symp. on Haptic Interfaces for Virtual Environments and Teleop. Systems, IEEE-VR2003, Los Angeles, 2003.

[6] Object Management Group, "Real-Time CORBA Specification, V 1.1," August 2002. ftp://ftp.omg.org/pub/docs/formal/02-08-02.pdf.

[7] Object Management Group, "Event Service Specification," 2001. ftp://ftp.omg.org/pub/docs/formal/01-03-01.pdf.

[8] Object Management Group, "Common Object Request Broker Architecture: Core Specification," 2002. ftp://ftp.omg.org/pub/docs/formal/02-12-02.pdf.

[9] Object Management Grp, "Naming Service Specification," 2002. ftp://ftp.omg.org/pub/docs/formal/02-09-02.pdf.

[10] Object Management Group, "CORBA® BASICS," 2003. http://www.omg.org/gettingstarted/corbafaq.htm.

0-7695-2112-6/04 $20.00 © 2004 **IEEE**

[11] Oliveira, M. et al., “The Pitfalls in System Design for Distributed Virtual Environments: A Case Study,” in Proc. of the Int'l Workshop on Immersive Telepresence (ITP 2002), Juan Les Pins, France, 2002.

[12] O'Ryan, C., Schmidt, D., and J.Noseworthy, “Patterns and Performance of a CORBA Event Service for Large-scale Distributed Interactive Simulations,” International Journal of Computer Systems Science and Engineering, 2001.

[13] Schmidt, D. C., “An Architectural Overview of the ACE Framework: A Case-study of Successful Cross-platform Systems Software Reuse,” USENIX Login Magazine: Special Issue on Tools, 1998.

[14] Schmidt, D. C. and Kuhns, F., “An Overview of the Real-time CORBA Specification,” IEEE Computer: Special Issue on Object-Oriented Real-time Distributed Computing, 2000.

[15] Schmidt, D. C., Levine, D., and Mungee, S., “The Design of the TAO Real-Time Object Request Broker,” Computer Communications Special Issue on Building Quality of Service into Distributed Systems, vol. 21:4, 1998.

[16] Vahora, F., Temkin, B., Krummel, T. M., and Gorman, P. J., “Development of Real -Time Virtual Reality Haptic Application: Real-Time Issues,” in Proc. of 12th IEEE Symposium on Computer-Based Medical Systems - CBMS 1999, pp. 290-295, 1999.

Proceedings of the 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS’04)
0-7695-2112-6/04 $20.00 © 2004 **IEEE**