In the Proceedings of the ASME Dynamic Systems and Control Division:
1999 International Mechanical Engineering Congress and Exposition
Eighth Annual Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems
November 14-19, 1999, Nashville, TN

AN ARCHITECTURE FOR HAPTIC CONTROL OF MEDIA

Karon E. MacLean, Scott S. Snibbe and Robert S. Shaw

Interval Research Corporation
1801-C Page Mill Road
Palo Alto, CA 94304
{maclean, snibbe, shaw}@interval.com

ABSTRACT

Most haptic interfaces are designed with a desire to produce simultaneous perception across multiple senses, with haptic behaviors that complement the task at hand. However, software architectures for haptic interfaces often distribute a virtual model and tasks among multiple processes and CPUs. This functional segregation can reduce the quality of haptic experience, by hiding important details of the dynamic model and reducing the communication bandwidth between the haptic controller and the application.

We present a means of closely coordinating a haptic interface with a virtual model, application content and other sensory displays. Our architecture features the ability to customize haptic behavior at a low level, in the dynamic relations among control parameters. Furthermore, we have located significant application control within the haptic process in order to facilitate effective bidirectional communication.

We describe some applications for haptically manipulating dynamic media; and propose extensions to our architecture for increased generality, decreased code size and high-level programmability for efficient application development.

1 INTRODUCTION

When a manual interface is used to manipulate a computer model or database, tight coupling between the manual interface and the model strengthens a user's control over his task. Active haptic interfaces provide an opportunity to enhance interaction beyond what is possible with passive mechanical controls such as a computer mouse or joystick. They can be used both to display and control data, and adjust their behavior to reflect or transmit context changes. However, to take full advantage of this input/output capability, we have found that certain criteria in the software architecture must be satisfied. These relate to the location of functional elements within software modules, communication between the modules, and the ability to craft and customize low-level details of the haptic control code. Here we propose a software architecture whose departure from common practice is aimed at haptic media control tasks. For the purposes of this paper, we will define *media* as stored or live dynamic content primarily in the form of digital audio and video. Our manipulation of media moves beyond literal representation to use an abstract or metaphorical physical model as the virtual model, placed between the user and the medium.

This paper will begin with a discussion of the pros and cons of current conventions in software architecture and hardware configuration. We then describe the key features and implementation of a modified architecture that alleviates constraints imposed by existing systems for tightly coupled media applications. The paper concludes with some application examples employing this architecture and a discussion of future work.

1.1 Function Segregation in Haptic Applications

Multiple tasks must be accomplished in a haptic interface application. For example, a typical set might be:

- Haptic device control (sensing and actuation)
- Virtual model definition, update and transitions
- Update of other sensory displays, e.g. visual or audio
- Monitoring of mouse and keyboard input.

It has become commonplace to separate functions into computational threads and sometimes CPUs, to accommodate different update rates, distribute compute load and optimize type of computation. For example, a graphics process might run on an SGI workstation while haptic control is handled by a realtime OS [11]. Such functionally separated threads are a practical way to access adequate processing power, but give rise to new questions: how to tightly couple threads, and to distribute tasks among threads.

A functional element critical to our discussion is the *virtual model* (VM), through which haptic interfaces typically communicate with the rest of the computer program. The VM

might be a literal description of a model or database entity, such as a geometric structure. The VM can also represent an abstract construct mediating user operations on the data – for example a physical metaphor such as a flywheel for editing a stream of digital video or audio media.

Tight Perceptual Coupling

By *tightly coupled* inter-module communication or perceptual simultaneity, we mean that when a user moves the haptic I/O device, the virtual model and other sensory displays react simultaneously, to human perceptual limits. Known perceptual time constants translate to minimum display refresh or sample rates (30 Hz for vision, 500-1000 Hz for touch [5]) and synchronization latencies (10 msec for haptic-audio simultaneity [4]). When controlling digital audio, for example, a haptically commanded transition must cause the corresponding audio event within 10 msec to appear simultaneous.

Location of the Virtual Model

When control of the sensory interfaces is separated, the location of the VM comes into question. Options include splitting, duplicating or isolating the model within a single process. A process that *doesn't* contain the VM may receive state updates with a fixed or variable delay. This is especially true if its update rate exceeds that of the VM process.

In some cases, the location of the VM is constrained. When the task is to directly manipulate a literal representation of a complex data model, it might be necessary for the VM to live in a graphics process since much of its content is needed by the graphics display but not by the haptic interface. Moreover, the haptic process will generally have the highest update rate in the system – when the VM is complex, updating it at the haptic rate may impose an excessive computational load.

Unfortunately, these constraints can make the achievement of tight perceptual coupling challenging, if the VM changes have a time constant smaller than the haptic-to-visual communication latency. For example, if the VM is updated at 30 Hz and the haptics process at 1 kHz, the haptics process may wait up to (33-1)=32 msec, which is haptically perceptible. Interpolation or filtering can help to reduce artifacts [14, 16] but the haptic process inevitably uses old data and will be less responsive.

This paper addresses a less-studied class of applications where it is natural for model detail to reside in the haptic process, by virtue of its relative simplicity and principal relevance to the haptic interface function. By managing an efficient VM in the fastest-running process and sharing state data with the slower processes as they require it, latency can be maintained below perceptual levels in all sensory domains and the goal of tight perceptual coupling accomplished.

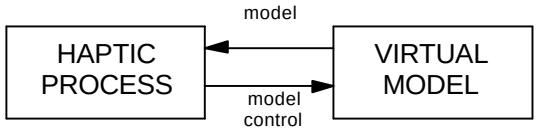

Figure 1: Bidirectional haptic communication with a VM.

Bidirectionality: Haptic Media Control

An interface *display* reveals information such as system state and content to a user – for example, a graphical display allows one to visualize a rendering of a computer model. An interface *controller* permits manipulation and control of information – e.g., a computer mouse is used to select or move elements seen on the visual display. A haptic interface is said to be *bidirectional* if it can be used as both a display and a controller (Figure 1).

Prevalence of Unidirectional Haptic Control

However, many haptic interface applications are dominated by the medium's display affordance rather than by its control capability, using haptic feedback as a rendering tool to provide another view of a virtual object. In these cases, the haptic action does not affect the model or other sensory displays, aside from cursor position. We will call such an architecture *haptically unidirectional* in its communication between the haptic process and a procedurally separate VM. The prevalence of haptically unidirectional architectures has influenced what the field has learned about using haptic interfaces (both psychophysical and algorithmic), and the software architectures and computer configurations devised to control and connect them to their applications.

There are other examples of haptically bidirectional architectures: for example, any VM that includes a compliant element, and haptic probing of dynamic VMs which makes the VM move [11, 16]. But to our knowledge, this work has focussed on *direct* interactions with and rendering of complex data models. That is, the VM and the data being manipulated are similar in form and directly coupled: one feels the virtual biological membrane being pierced, or the three-dimensional virtual shape shown on the screen.

Haptic Media Control

It is also possible to manipulate and operate on media, *indirectly*, mediated by a VM that has an abstract and tool-like relation to the data set. By indirect, we imply not laxity in either the haptic-VM or VM-media couplings, but the abstract relation of the VM to the media. This kind of VM is felt, but not necessarily seen. For example, a metaphor of turning a wheel might be used to haptically browse a stream of video frames [9]. The user holds a physical wheel, imbued with virtual dynamics that aid in controlling the video stream; but he sees the video frames, not an image of a turning wheel.

We refer to this operational, manipulative type of interaction as *haptic media control*.

An attribute of this type of haptic control is the inherent simplicity of the VM. Since the VM's function is now purely to enable haptic interaction, rather than to represent every feature of, say, a complex 3D data set intended for graphical display, it need not contain details extraneous to the haptic simulation. Because it is perused serially, updating even a multi-degree-of-freedom haptic model will generally be less involved than a graphical display.

Thus, in the case of haptic media control, it is both critical and possible for the haptic process to have a tight bidirectional connection to the VM, by including the VM within the haptic process.

1.2 Crafted Code for Haptic Expressiveness

The success of some haptic applications relies in no small part on the interaction's aesthetic and expressive qualities. We believe that attention to these details maximizes a user's sense of connection to and control over the media, increasing the value added by haptic feedback. This requires that the application designer have access to low-level details of the dynamic model used for haptic feedback.

Any haptic display built today and in the near future has a constrained palette compared to the physical world we feel with our bare hands. Whether the goal is to present a rich, pleasing sense of presence, facilitate manual control or increase efficiency in performing a specific task, the designer needs to use the palette well. He should be able to extend it creatively, and obtain the full benefit of the high local haptic update rates that many systems can now achieve.

This ability is limited in commercially available software tools that require a designer to download a packaged virtual model to a local haptic CPU, then update its parameters at a rate slower than the haptic update period. While some of these packages expedite the job of prototyping feels through quick parameter iteration, this limitation means the designer can't try out new low-level models and algorithms which aren't included in the tool library, or respond to haptic events or model transitions at haptic perceptual rates.

2 CONFIGURATION & ARCHITECTURE: CURRENT CONVENTIONS

Having described what we consider critical features for a haptic control architecture (tight bidirectional coupling between haptic process and virtual model, exploited by designer access to low level haptic code), we offer an overview of the software architectures and computer configurations in common use. First we consider the functional division of the complete application code; then to the way computational units are utilized and connected.

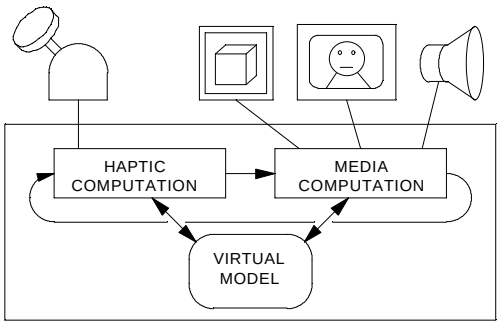

Figure 2: Single-threaded architecture

While this paper deals principally with software, hardware trends are relevant in the opportunities and pressures they present to software developers.

These observations are not exhaustive, and we have omitted specific attribution when we consider the subject evolved communal property. The point is that despite promising forays in other directions, the community as a whole is choosing a constraining set of architecture models with few and diminishing exceptions.

2.1 Software Architectures

In this context, we use the term *software architecture* to describe the distribution of virtual model update and other tasks among CPUs and processes, and the manner and rate of synchronization and data sharing. All of these examples assume adequate realtime performance of the host operating system; we will not cover OS issues here.

Single Process

The most straightforward and venerable method of integrating haptic control with content control is for both functions to reside in a single-threaded program (Figure 2). The principle advantage of this approach is the ready availability of data to all parts of the program. Latency issues that relate to inter-module communication are avoided.

Multiple update rates can be achieved through timer interrupts or interleaved functions (e.g. execute haptics function five times, then graphics function one time). Both approaches effectively put the burden of scheduling on the programmer, and while the result can give good realtime performance, development effort becomes excessive for any but the simplest systems.

Multiple Processes or Programs

The logical next step is to segregate functions with dramatically different requirements into separate and appropriate processes or programs (Figure 3). The processes may have independent clocking rates and their own data sets. Depending on the affordance of the OS (in the beginning there was DOS, and it is still in use), mechanisms for

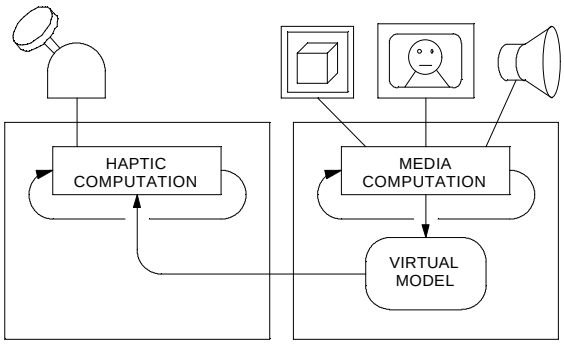

Figure 3: Multi-threaded architecture

interprocess communication (IPC) include local variables, shared memory, sockets, message passing and realtime clocks. The functions may also be executed by independent *programs* that communicate in similar ways; this method is particularly common when multiple CPUs are used.

The virtual model must now either be located in just one process, or partially duplicated. There have been numerous approaches to splitting and locating models and in choices of the information to be shared between the high-speed haptic control function and the rest of the system. Each approach has its own design priorities and constraints, and varying degrees of success have been achieved.

2.2 Hardware Configurations

The three computer hardware configurations described here loosely cover most possibilities for computer-controlled haptic feedback.

Single CPU

All application software resides on a single CPU. The advantages of this approach are simplicity of development, sometimes cost, and ease of fast interprocess communication.

The primary limitation of this configuration is that servo rates of the various functions are not independent as computational bandwidth is approached. Further, different operating systems and computer architectures are more suitable for different functions. For example, an SGI running IRIX might be desirable for graphic performance, but is not ideal for realtime low latency hardware control.

Although the speedup and convergence of computer hardware platforms suggest that more is possible in this mode, the dominant Wintel operating system does not show signs of becoming a more hospitable host to quality haptic control. Even as computers become more powerful, limits are inevitably pushed. Thus, a choice of system and priority of update must often be made on the basis of the dominant concern (e.g. graphics display) and the performance of the other functions will consequently suffer.

Multiple Networked CPUs

To solve the problem of computationally hungry processes starving others and to allow optimization of the host OS to the thread's function, multiple CPUs can be networked together. Each hosts one or more processes or programs, and thus requires a multi-threaded software architecture. The communication, usually bidirectional, has most often been implemented using sockets over Ethernet protocols [10, 11] or serial links. The latencies inherent to shared network paths make serial and parallel links an increasingly attractive option when the CPUs are physically nearby.

For the same CPU power, this configuration allows higher and more consistent local rates and more complex models, but the price is paid in a finite synchronization rate and a consequent degradation of perceptual tightness. For example, an artifact might be introduced to the haptic process at the connection rate, often 30 - 100 Hz.

Low Cost Embedded Haptic Controller

When the goal is to build a lower cost or physically diminutive haptic interface, the obvious hardware configuration is a variant on the multiple-CPU configuration: replacing the haptic control computer with a dedicated and relatively low-cost microprocessor [2, 7].

While this is an attractively efficient approach which will become more prevalent as the price/power ratio of embeddable CPUs drops, it is especially prone to certain shortcomings. Factors of size, cost and capability of the haptic control CPU may dictate aspects of the communication protocol. There might only be enough bandwidth for a small amount of information to flow, and perhaps only in one direction. Thus, artifacts such as buzzing and loss of perceptual synchronization are introduced, as described for the multiple-CPU configuration.

Even more relevant to our design priorities, the more limited nature of software development tools for embedded CPUs and the unfamiliarity of application designers with embedded code development makes the pre-packaging of code modules attractive to many users of commercially available systems.

2.3 Non-Conventional Approaches: Related Research

We are not the first to object to 100 Hz refresh jitter, or to choose to do something about it. A few efforts recounted here substantiate our own belief that the problem is both important and solvable. These prior solutions have been either presented incidentally, or not made general enough for adoption by the community. We are striving to highlight the importance of such architectures and generalize for contemporary and future haptic programming needs.

Those who define the haptic Holy Grail as smooth perceptual synchronization with a virtual model have often been

associated with dynamic modeling. Perhaps the most pointed observation of synchronization artifacts and effective solution is offered by Vedula [16]. In this case, attention to detail in dynamic computer models led to dissatisfaction with low fidelity in the haptic display. Using two CPUs connected over Ethernet, Vedula created a more sophisticated local model in the haptic controller such that slower-than-desirable synchronization updates could be spread over the entire update period. Gillespie [1] addresses many of the issues inherent in precise simulation of complex mechanical systems, including discretization instability and dynamic coupling between manipulandum and virtual model. A single-CPU configuration and a dynamic model updated with every haptic servo provides high fidelity haptic feedback in a piano key simulation.

Others using single-CPU configurations have focussed on haptically smooth presentation of virtual models, dynamic or static, whose spatial resolution is low enough to perceive haptically – a different but related problem. These include the constraint-based methods of [11, 17] and force shading [8].

More a historical note than a comprehensive solution, the Montage commercial non-linear video editing system was notable for its originality at the time [7]. One of the first implementations of embedded control of haptic feedback, it included a haptic browse-wheel which reconfigured the number of detents per rotation depending on whether film or video footage was being edited. Montage's exact architecture for haptic control is unknown. We believe it was a one-way link from haptic display to media with a dedicated processor whose behavior could be changed by downloading different position/force tables.

2.4 Implications

Modularization and separation of components is the dominant methodology in haptic software architecture design. While this approach is an effective solution for one problem, it is currently linked to a movement towards generic low level haptic control utilities that tend to isolate the application builder from crafting and individualization of the haptic experience. One of the primary factors in this evolution is in many respects a positive one: the availability of software packages which allow a relative novice to write code for a haptic device, without requiring expertise in robotic control or hardware design [3, 12].

Today’s prevalent haptic architectures have been driven by the real need to develop simple, higher level control functions while ensuring safety. These control functions rely on models residing within the haptic controller, which are periodically updated through lower-frequency calls from the user’s code. Introducing open-loop execution between interprocess updates reduces the quality of haptic feedback, and often introduces unwanted artifacts. Sealing off the model within the embedded controller and preventing the user from modifying the closed-loop haptic code compromises the diversity of expression inherently possible with the device. Such models broaden the market acceptance of haptic devices initially, but in the longer term may reduce quality and creative sophistication in applications.

As proponents of highest quality haptic force feedback, we argue for allowing the casual as well as expert user to experiment creatively with the potential of haptic feedback by manipulating both the dynamic model and its parameters.

3 NEW ARCHITECTURE MODEL

While constrained APIs are neither theoretically necessary nor inevitable for any of these hardware configurations, they have obvious advantages. We propose an alternate approach which retains these features while alleviating their drawbacks, applicable to a particular class of applications – those where the virtual model is simple enough to be updated at haptic control rates. We predict that this class will increase in importance due to two factors: faster cheap, embeddable CPUs, and the spread of haptic media control applications.

The key difference in our architecture is in the division and coordination of the code: we let the haptic process drive the model, and use a sufficiently sophisticated local haptic model such that the discretization of model synchronization is not perceptible. We argue that the only necessary tradeoff is extra care in the system design to prioritize coupling and customization of low level haptic code. Further, we maintain that such architectures can migrate into commercial general-purpose software packages.

Designed for the tightly coupled haptic manipulation of continuously streaming media such as audio and video, our architecture had the following requirements:

- Rich, dynamically configurable haptic behaviors.
- Functionally separate control of the haptic interface, multiple types of media and standard input/output such as the keyboard and mouse.
- Tight integration of haptic motion with the manipulated media, providing the sensation of direct control.

Up to a point, these constraints are respected by prior systems we have developed, as well as many found in the general research community and industry. However, our desire to integrate the haptic interface more closely with continuous media, to dynamically configure behaviors and to allow non-expert users to prototype haptic behaviors led us to a significant variation on this prior work. The architecture has the following key elements:

- Bulk of computation on the haptics side of the model for very tight feedback with user.
- Multiple processes running at different refresh rates.

- Two-way communication via shared memory, message passing and remote serial/Ethernet protocols.

The first point is where our model departs from most current convention. Other features of our architecture include run-time targeting to different or multiple haptic devices, cross-platform development and a simple callback model for registering haptic and non-haptic processes and establishing the shared memory and other communication protocols. The callback model allows the authors of haptic experiences to focus solely on the haptic coding and ignore the details of timing, filtering and other computation.

3.1 Control Weighted towards Haptics Process

Haptics Process as Master

Many multi-threaded architectures have employed the haptics process as a slave, responding to commands from a master process [2, 3, 12]. In our media control applications, we have found that exactly the opposite is desirable: the haptics process runs at the highest refresh rate and must be immediately responsive to the user and to the media. It should both drive the VM update and derive its behavior from the most current representation in the system. We have found the absence of this balance obstructive in our attempts to use off-the-shelf haptics software toolkits.

We have therefore located the virtual model in the haptic process, and made it responsible for driving or coordinating events in the other processes. Since the VM for our type of application is usually a simple mechanical system, this does not impose a computational strain even in single-CPU configurations.

Option to Customize Haptic Code

While API modules are invaluable for streamlining standard tasks, access to low level haptic code is also desirable from several standpoints. For example, with their different refresh rates, the haptic and media experiences must be mediated: e.g., the high resolution haptic state information must be integrated or otherwise analyzed to relate correctly to the visual or audio media displays. Conversely, low refresh rate information from the media can be interpolated or filtered in the haptic process. We have found it essential to directly explore the many methods for accomplishing these tasks in a given context.

As to the haptic behavior itself, it is the freedom to experiment with what happens in between interprocess updates that we wish to preserve. For instance, we have found that a haptic behavior may be enhanced if its parameters are updated at the higher haptic refresh rate, and this necessitates custom programming within the haptic servo loop. In some cases this behavior consists of triggering a haptic transition or a non-haptic event such as a sound. In others, the event is a switch to a new continuous model.

Any of these customizations might later be packaged into an API model for easy re-use, but they can't be created without the ability to prototype completely new behaviors that are not combinations of existing models.

Therefore, though we modularized our architecture into a framework for IPC and low-level functions such as velocity signal filtering and gainset choice, we allow any of these to be easily over-ridden with custom code. We also make it easy for a programmer to completely compose just the critical parts of the haptic process by registering callback routines.

3.2 Multiple Processes

We subdivide our code into separate processes or threads. Each process runs at either a continuous, guaranteed refresh rate, or as an event-driven blocking process. One instance of our architecture uses two different computers, one for dedicated control of haptics and a second for the control of digital video. A Pentium desktop computer running QNX hosts the haptics control process, but these functions are intended eventually for an embedded processor. In this particular example, the following processes run at all times (Figure 4).

Haptic Servo Loop. The haptic process updates at 1Khz and continuously measures the position of the haptic device, computes filtered velocity and position values and outputs a force derived from the current virtual model representing the media.

Media Client. Delivers and receives messages from the Media Server which may be another process on this machine, a dedicated device external to the Haptics processor, or a

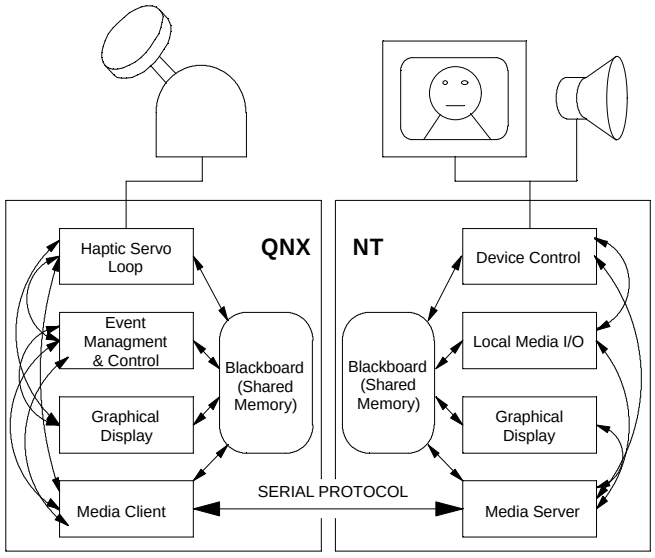

Figure 4: A two-machine instantiation of our architecture for haptic control of video. Interprocess messages are shown as curved lines; references to shared memory as straight lines.

separate computer with digital media capabilities. This process updates at twice the refresh rate of the digital media (60Hz), to guarantee a minimum latency of one video frame.

Event Management and Control. A central, blocking process to handle scheduling and termination of the application and general I/O (keyboard management, mouse, etc.).

Graphical Display. A local graphical display is sometimes used for debugging or additional high bandwidth visual feedback. This process runs continuously at the refresh rate of the display device (60Hz). In an embedded system, this process would most likely not be running.

The following processes run on a separate machine in the given example, but conceptually could run on the same machine, or be encapsulated into a dedicated device:

Media Server. Responds to Media Client messages and controls the presentation and manipulation of digital media (video and audio), primarily by sending messages to the Device Control process. The Media Server updates at twice the media refresh rate (60Hz).

Media Device Control. Handles the details of controlling the media devices and responds to requests. This process is blocking and event-driven.

Local Media I/O. A dedicated process to respond to local control of the media device; it is blocking and event-driven.

3.3 Inter-Process Communication

Processes communicate via three different protocols:

- A shared memory blackboard, semaphore protected, for maintaining the haptic model and other information on the state of the haptic device, such as filtered velocity and position.
- Inter-process messages, primarily for event notification and overriding control.
- Serial, parallel or Ethernet remote connection to transparently enable control of remote processes.

The goal in the use of these protocols is bidirectional communication between the haptic process and the controlled media: each interface (haptic, visual, audio) both sends and receives data. There are two closed servo loops in this system: the local fast haptic loop, which includes the haptic display and the user's hand, and the slower outer loop which encompasses the media being controlled (Figure 5). From this perspective, the haptic interface is much more than a "display" – it is a nonlinear block in the system diagram, closely affecting downstream events. Likewise, the closing of the outer media loop provides the haptic loop with context information that in turn alters the haptic behavior.

Each of the lines connecting the block elements in Figure 5 are realized through one of the forms of IPC listed above. In many of our applications the Haptic Servo Loop and Media

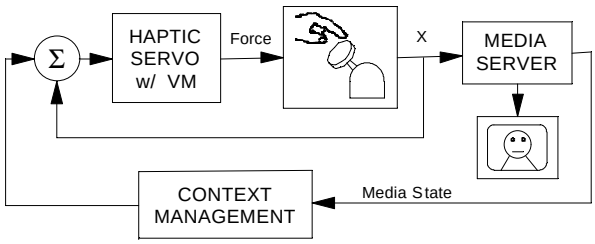

Figure 5: Inner and outer closed control loops.

Client processes are on separate CPUs and their bidirectional communication is transmitted through a serial connection. The Media Server coordinates the serial communication from the haptic processor side, and stores the Media data in a semaphore-protected shared memory receptacle for access by the Haptic Servo Loop. The Event Management and Control process intercepts special keyboard commands such as stop, pause, reset, and sends messages directly to other affected processes.

While a closed-loop approach is familiar to the robotics field, it is alien to the traditional model of media control. The prior work in the Montage system hinted at its need – the behavior of the haptic device was contextually sensitive to the state of the medium. For our more sophisticated media browsing experiments, we require a higher bandwidth context sensitivity that provides strongly enhanced value in comparison to passive haptic feedback devices.

3.4 Details of Implementation

Our architecture could be applied to any of the hardware configurations mentioned in Section 2.2. For the video applications described below (Sections 4.1 and 4.2), it is implemented using two serially connected Pentium computers. One runs QNX and handles the haptic and media client processing. The other runs Windows NT and has a Digital Disk Recorder installed to provide full-frame random access video support. Custom software on the NT side runs the Media Server and utility processes. A custom serial protocol is used to communicate between the two machines. We have found that QNX provides a dependable real-time architecture for our experiments. Such experiments have proven difficult or had unreliable results in other UNIX architectures or within Windows or MacOS.

The application example Aladdin (Section 4.3) is implemented on a single Pentium computer, since both haptic and audio processes run under QNX. Processes communicate via shared memory and message passing, and are synchronized at 80-100 Hz. The haptic and audio processes can be run on separate CPUs with no loss of performance if interprocess communication rates of 80 Hz can be maintained.

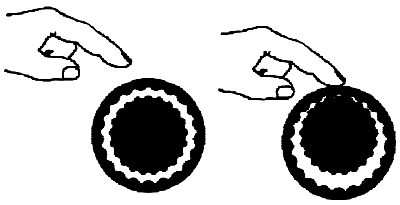

Figure 6: Haptic Clutching. A clutch metaphor is used to connect a virtual inner wheel with the physical outer wheel. Orthogonal force is sensed on the physical device, and the user's pressing on the orthogonal axis causes the outer wheel to engage the inner wheel. The user imparts momentum to the inner wheel by engaging and releasing it; and feels relative motion between the two wheels as bumps sliding by.

4 EXAMPLE APPLICATIONS

To establish the value of our architecture, we describe three example applications that would be difficult to realize with the same quality and responsiveness using other systems. The first two applications are for the browsing of video, while the third is the use of a haptic device for delivering ambient information about a room.

4.1 Haptic Clutching

One important metaphor we have been exploring in our research is the engagement of a simple low-DOF haptic device with a more complex virtual model. One example of this is the *haptic clutch*, where we simulate the clutched engagement of a concentric pair of virtual single-DF wheels (Figure 6) with the help of force sensed on the axis orthogonal to the single real wheel's rotation [13, 15]. The inner and outer virtual wheels are modeled as inertial elements with bumps on their facing surfaces, which correspond to features in the media. The wheels couple when the bumps mesh, with the manipulated wheel acting as the outer driver.

A user can interact with this virtual dynamic system in several ways. If he pushes down with a firm force, the two wheels engage as a single rigid body that the user can rotate it in either direction. We have linked the rotation of the inner wheel to the advance of frames in digital video, providing a means to directly shuttle between frames. If the user relaxes his downward force, however, the inner wheel is released and continues to spin with the imparted velocity. The video advances with the speed of this virtual inner flywheel, which is continuously variable from slow advance to extreme fast-forward. The user can continue to push down and shove the inner wheel to increase the media browsing rate. By pushing down steadily, the user can brake the inner wheel, with a satisfying slip as bumps fly by at slower and slower speeds. The physicality of the medium is restored with this interface – to stop the video requires the user to exert force and dissipate the flywheel's momentum.

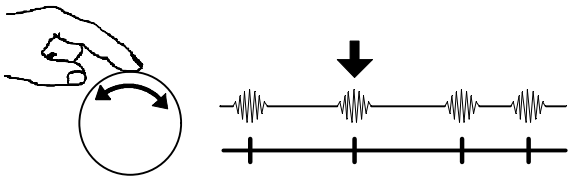

Figure 7: Foreshadowing. A piece of media with marked temporal regions can be browsed using our foreshadowing metaphor. The wheel acts like a normal video shuttle wheel, but as the user approaches marks in the media, a texture is overlaid on top of the wheel's motion. Users can also add marks by firmly pressing. The marks gradually rise and lower around the point of interest, alerting the viewer to the upcoming event before it is actually reached.

This behavior is described by the following dynamic system. These equations use rectilinear rather than rotational coordinates for simplicity, and the kinematic state of the virtual outer wheel corresponds to the measured state of the real wheel.

$$F_{clutch} = f_{\perp} h \sin(x_o - x_i)$$
$$0 = M_i \ddot{x}_i - F_{clutch}$$
$$F_{act} = B_o \dot{x}_o - F_{clutch}$$

The first relation defines F_{clutch}, the force transmitted between the two virtual wheels. $f_{\perp}$ is derived from the measured orthogonal applied force, used to indicate the degree of engagement with the virtual inner wheel. h is a dimensionless constant indicating the height of the bumps. $\vec{x}_o$ and $\vec{x}_i$ are the measured state of the real wheel and the computed position of the inner wheel, respectively; M_i and B_o are the mass and damping of the two virtual elements. F_{act} is the force applied via the 1-df actuator.

Despite its simplicity, it would be hard to implement this system with current commercial haptic APIs, because the virtual model's state must be communicated to the media server faster than interprocess communication rates. There is a large class of similarly simple dynamic systems that can be used to create experiences beyond direct haptic metaphor or representation.

4.2 Foreshadowing

An example that shows the need for bidirectional communication is haptic foreshadowing for video (Figure 7). In browsing and annotating video footage, we wished to indicate important areas of the footage with a haptic cue. This function is commonly required in editing tasks and is normally accomplished with visual cues along a timeline.

Figure 8: The Aladdin haptic door knob, which combines force feedback with a thermal display.

We tried several techniques for haptic annotation and found that a texture with gradually increasing and decreasing magnitude worked well to "foreshadow" an important mark before it was reached. The wheel behaves like a standard spring-centered video shuttle wheel, with this texture overlaid as left/right rotational noise of varying frequency, depending on the distance to the mark. At the same time as the user is browsing the footage, they are also able to make additional marks by firmly pressing down to on the wheel, equipped with a force sensor. Thus, the haptic process must request and deliver information to the media process bidirectionally and at a rate that may exceed the visual media refresh rate, depending on the speed of browsing. Further, the haptic behavior is one that isn't part of a supported application toolkit, requiring custom coding. In developing this application we tested several standard and non-standard behaviors including nudging, buzzing and modifying viscosity and friction. This would have been difficult with existing commercial packages.

4.3 Aladdin: an Expressive Haptic Door Knob

The project Aladdin is an experiment in creating and using an ambient haptic object, integrated with an auditory display. A normal-appearing door has a haptically active knob that senses knob motion and touch while displaying force and temperature (Figure 8). Audio outputs, coordinated by an audio process, are synchronized with the driving haptic process using small audio buffers to ensure agility of haptic-audio coordination [6].

One potentially useful application is the delivery of information about the space on the other side of the door, or of a room at a remote site. In this case the media is the information fed from sensors within the room. For example, haptic textures, force and temperature can be sculpted into knob behaviors that imply properties such as the number of occupants, their mood, level of activity and desire for privacy.

Another ongoing experiment is to study the integration of multiple modes of sensory feedback into a single display, mimicking rich multisensory experiences in the natural world. Aladdin assumes behaviors with expressiveness dependent on both feel and sound. The tight perceptual coupling allowed by this architecture is necessary to create the illusion of a presence in the door, and to allow a user to control that presence with the knob. Likewise, we continually extend our expressive palette by crafting new low-level haptic feels, and weaving than together with crafted sounds. Subtle adjustments to the haptic code make the difference between a compelling experience and a dull one.

5 FUTURE WORK

Our next steps relate to accommodating the diversity and power of new embedded processors and the haptic applications they enable; and to making our architecture more accessible to novice (or hurried) programmers.

5.1 Embedding Haptic Displays

We have proposed an alternative to the current prevalent view of a haptic interface as a slave to a desktop PC, whereby the haptic interface becomes the locus of control and arbitration for a distributed, networked system. In a more general instantiation of our architecture, one can imagine a system that includes a haptic interface and its local embedded controller configured as a collection of peer appliances connected through serial or network links (Figure 9). Potential peer devices vary both in power and function of local host. A single haptic interface might drive multiple peer targets, depending on the context, or it might be specific to one.

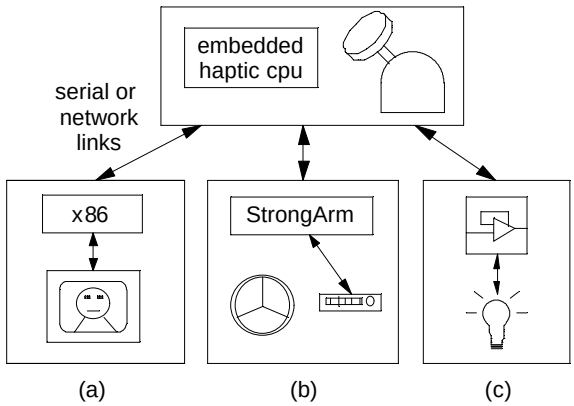

Figure 9: Embedded haptic interfaces can control arbitrary targets such as (a) an embedded x86 driving a video monitor; (b) a StrongArm controlling an automobile sound system; (c) a network of analog circuits, each responsible for one light in a room lighting system.

This view of haptic control is both more localized than what we have seen proposed or implemented elsewhere, and capable of arbitrary degrees of interconnectedness with the rest of the world. Haptic displays hereby become distributed local control elements, part of larger interlinked I/O systems, rather than a slave to a variant of a desktop PC.

5.2 Higher Level Authoring

Our proposed architecture presents an environment where the authors of haptic phenomena must have intimate knowledge of the discrete implementation of differential equations and other low-level physics. The reconfiguration commands coming from the other processor consist of downloading code-fragments or commands to engage custom routines. We believe that this level of authoring is essential to the crafting of high-quality, non-trivial and compelling haptic applications. However, we are also working to find ways of authoring such experiences at a higher level.

Our current research is looking at four different levels for authoring without recourse to hand coding. The first level is a software toolkit of chainable routines and utility functions to encapsulate common controls and filters. The next level is a scripting language for specifying the connection of these components with a simplified language. The third level is the use of equations, rather than code, which are interpreted or compiled on the fly to control the haptic behavior. Finally, a graphical authoring environment that is analogous to multimedia or music sequencing programs should one day be possible. Before that day, however, we need to determine how to transition, layer, combine and compose haptic phenomena. A strong component on the road to this system includes psychophysical experimentation to determine the "basis functions" and degrees of freedom expressible with particular haptic devices.

ACKNOWLEDGMENTS

Other members of Interval's haptic research group have contributed to the evolution of these ideas: Jayne Roderick helped create Aladdin and did much of the architecture problem-solving. Bill Verplank continually reminds us that haptic expressiveness is realized at the most fundamental levels. Jesse Dorogusker has made major contributions to our hardware design and execution.

REFERENCES

[1] B. Gillespie and M. Cutkosky, "Interactive Dynamics with Haptic Display," presented at the 2nd Ann. Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems, ASME/WAM, New Orleans, LA, DSC:55-1, pp. 65-72, 1993.

[2] Haptic Technologies, *PenCAT™ High Performance Haptic Pen*. Montreal, Canada, 1998.

[3] Immersion Corporation, *I-FORCE™ 2.0 SDK Programmer's Reference Manual Version 2.0*. San Jose, CA, 1998.

[4] D. J. Levitin, M. V. Mathews, K. MacLean, L. Chu, and E. Jensen, "The perception of cross-modal simultaneity," in preparation, 1999.

[5] K. E. MacLean, *Emulation of Haptic Feedback for Manual Interfaces*, Ph.D. Thesis, MIT, February 1996.

[6] K. E. MacLean and J. B. Roderick, "Aladdin: Exploring Language with a Haptic Door Knob," presented at the 12th Ann. ACM Symp. on User Interface Software and Technology (UIST'99), Asheville NC, , 1999.

[7] Montage Inc., *The Montage Nonlinear Editing System*. New York, 1992.

[8] H. B. Morganbesser and M. A. Srinivasan, "Force shading for shape perception in haptic virtual environments," presented at the 5th Ann. Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems, ASME/IMECE, Atlanta, GA, DSC:58, 1996.

[9] M. Naimark, "Field Cinematography for Virtual Reality Applications," presented at *VSMM '98*, 4th Int'l Conf. on Virtual Systems and Multimedia (invited paper), Gifu, Japan, , pp. 23-32, 1998.

[10] W. Plesniak and J. Underkoffler, "SPI Haptics Library," presented at the 1st Phantom User's Group Workshop, Dedham, MA, MIT AI Lab TR 1596, 1996.

[11] D. D. Ruspini, K. Kolarov, and O. Khatib, "The Haptic Display of Complex Graphical Environments," presented at SIGGRAPH '97, Los Angeles, CA, , pp. 345-352, 1997.

[12] Sensable Technologies, *The GHOST™ SDK Version 2 Programmers Guide*, 1998.

[13] R. Shaw, S. S. Snibbe, W. Verplank, K. E. MacLean, and J. Dorogusker, "Orthogonal Force Sensing in Haptic Interfaces," Interval Research Corp., Palo Alto, TR IRC #1999-070 (in preparation), 1999.

[14] S. S. Snibbe, S. Anderson, and B. Verplank, "Springs and Constraints for 3D Drawing," presented at the 3rd Phantom Users' Group, Dedham, MA, MIT AI Lab TR #1643, 1998.

[15] S. S. Snibbe, R. Shaw, K. E. MacLean, J. B. Roderick, K. Johnson, O. Bayley, and W. Verplank, "Haptic Metaphors for Digital Media," Interval Research Corp., Palo Alto, TR IRC #1999-071 (in preparation), 1999.

[16] S. Vedula, "Force Feedback in Interactive Dynamic Simulation," presented at the 1st Phantom User's Group Workshop, Dedham, MA, MIT AI Lab TR#1596, 1996.

[17] C. B. Zilles and J. K. Salisbury, "A Constraint-based God-Object Method for Haptic Display," presented at the 3rd Ann. Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems, ASME/IMECE, Chicago, IL, DSC:55-1, pp. 146-150, 1994.