



Distributed Multimedia Systems R&D at CREATE

Since 1996, the UCSB Center for Research in Electronic Art Technology (CREATE) has been the home of a series of projects on distributed software systems for real-time and multimedia applications. Several aspects of our work are relevant to a new classes of applications as more and more systems are built using distributed object software technology for real-time services. This white paper describes our previous projects and innovations in this area and our plans for the future.

Background

For many years, computer technology has been driven by arts, entertainment and gaming applications. It is no accident that all common high-density storage media (e.g., CD, DAT, and DVD) were originally developed for media data storage, that the fastest processor available today is embedded in a gaming platform (the PlayStation 2), or that the most innovative user interfaces are those found on gaming and entertainment applications.

For every generation of computer hardware, there have been some computer applications that are outside of the range of performance that can be provided by off-the-shelf processors. The two traditional answers have been either to build supercomputers (i.e., fine-grained homogeneous parallel processing), or to build distributed systems (coarse-grained heterogeneous parallel processing). The “demanding” applications that are of interest at CREATE include multi-user virtual environments, many-channel spatial sound processing, and “orchestral-scale” sound synthesis. Our current integrated virtual environment system uses “best-effort real-time” distributed sensor-tracking, stereoptic visual rendering, complex sound synthesis and multichannel spatial processing, and object database services, all implemented in a heterogeneous network of off-the-shelf computers.

An increasing number of important application domains require real-time distributed object processing, however, including network application servers, multimedia content streaming, high-level simulations, computer graphics, and e-commerce.

The scope and reliability of our distributed multimedia applications is hindered by the fact that current real-time distributed object software frameworks (such as CORBA and Java RMI) lack two central features: (1) languages to describe the real-time behavior of complex distributed software components, and (2) tools for configuring and monitoring distributed real-time object-oriented software. Despite the ad-hoc solutions available from several sources, there is no general and efficient way to program large-scale distributed real-time object-oriented software. If we develop the infrastructure necessary to build this kind of system for our applications, software developers working in many other areas would also profit.

This white paper presents an historical overview of CREATE’s R&D on distributed systems, describes our recent developments, and describes what we feel are the most promising areas for future investigation in this area. We have several concrete projects in planning to continue our work in this domain.

Distributed object computing

Distributed real-time object-oriented software is a topic of much current interest; for example, the June, 2000 issue of the *IEEE Computer* magazine is a special issue on the subject. At CREATE, we have been involved in distributed software since 1995. The high-level goal of our effort is to provide flexible and scalable real-time distributed environments for applications that involve complex signal processing, multi-user virtual reality, large-scale simulations, distributed databases, and wide-area (and wireless) networks.

Distributed object computing (DOC) is the standard method of building scalable, high-volume or high-reliability computer applications. Systems built using such standards as the *Common Object Request Broker Architecture* (CORBA), can be used to build distributed systems. CORBA offers the developers of large software systems the potential of good scalability, platform and language independence, and legacy system and database interoperability. Given these developments, it is now possible to perform wide-area distributed processing with little of the programming overhead associated with traditional multi-tier applications. This advance in technology has led to a number of new application architectures, and to the widespread use of CORBA in many application domains, even areas where real-time interaction and controlled latency are required.

The R&D team at CREATE has been working to apply and extend CORBA for use in large-scale real-time applications. Our first major project (never fully funded) was called *Next-Generation Networked Multimedia* (NGNM, see the Web page <http://www.create.ucsb.edu/ngnm>). The high-level NGNM vision still serves as our road-map, and the projects we introduce below—IDOT (1997), DRIVE (1998), HPDM (1999,2000), ATON (2000/01), and DPE (2000/01)—can all be considered “children” of NGNM.

Our emphasis has always been on integrating off-the-shelf components (e.g., standard operating systems and CORBA ORBs) with specialized “glue” code and portable “light-weight services” to construct reasonable-performance scalable systems that provide “best-effort, predictable” (rather than guaranteed) real-time behavior for applications.

The IDOT Project: CORBA Interoperability and Performance

During 1996/7, the *Impact of Distributed Object Technology Using CORBA* (IDOT) Project developed a suite of programs in C++, Java, and Smalltalk to test the true interoperability and performance scalability of three commercial CORBA ORB software packages: OrbixWeb from Iona Software, VisiBroker from Visigenix/Borland, Inc., and DistributedSmalltalk from DNS Technologies. We ran extensive compatibility and performance tests

in Sun/Solaris, SGI/IRIX, and PC/NT systems. This project was also our first collaboration with the CalTrans Test-bed Center for Interoperability (TCFI) Lab.

The IDOT project report includes a comprehensive discussion of the practical use of the ORB products (focussing on real-time and data-streaming applications), and extensive code examples in three languages. We also evaluated the ORBs along several dimensions. Our finding was that the ORBs we tested indeed performed well in the interoperability and functionality testing, but that there were significant differences in terms of ease of installation and usage, provision of additional services, and maintain-ability.

We have distributed both the detailed IDOT reports and the source code for our entire test suite from the project’s Web site at <http://www.create.ucsb.edu/idot>. This project served as the basis for several later projects, such as ATON and HPDM (see below).

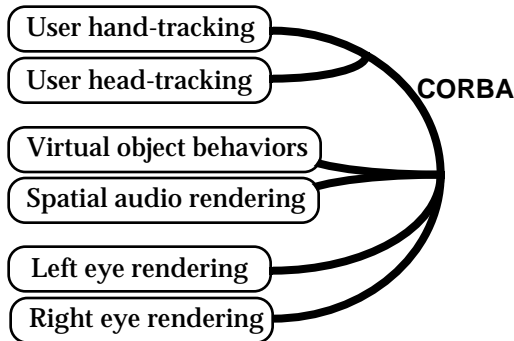
Distributed Virtual Environments

Virtual environment (VE) technology (also known as virtual reality, telepresence, or immersive user interfaces) is emerging as the basis of the next generation of multimedia user interfaces. Today, there are widely used virtual world description languages, and portable immersive rendering and interaction systems such as the *Distributed Real-time Interactive Virtual Environment* (DRIVE), and the *VR Utilities* (VRUT) systems.

The development of the DRIVE system has been motivated by the desire for a portable, scalable, and easily programmable immersive user interface system. The DRIVE architecture is based on a distributed, real-time, object database that represents the virtual “objects” in one or more virtual worlds. The database is connected to (visual and/or aural) renderers (one or more per user), user input trackers, and software applications that can change the state of objects in a world.

The DRIVE Project dealt with immersive user interfaces that use telepresence technology for a variety of applications. In phase 1 (1997), CREATE developers used several software packages for virtual world building and user navigation. In phase 2 (1998/9), we focused on world-building (using real-world

data such as aerial photographs), and applications in spatial navigation and multi-user interaction. Our work is to be portable among modeling languages and hardware platforms. We have documented this project and its deliverables on the Web at <http://www.create.ucsb.edu/drive>.



The CORBA Telepresence Architecture

Our recent work has been focused on issues related to delivery systems for complex VE systems. We have moved away from the earlier DRIVE infrastructure to a newer, framework using the highly-optimized VRUT renderer (developed at UCSB) and a CORBA-based distributed architecture as shown in the Figure on the previous page.

For the future, we're continuing to study world-building and VE delivery in the ATON project, and have published in-depth surveys on both of these topics.

The ATON Project: Real-Time CORBA for Telepresence Systems

The ATON project (1999-2001) is a collaboration between CREATE, the Computer Vision and Robotics Research Laboratory at the University of California, San Diego, and the CalTrans Test-Bed Center For Interoperability (TCFI). The project involves topics as diverse as robotics, computer vision, distributed multimedia processing, and VEs.

The project partners felt that a telepresence-based user interface paradigm would be of great use to the Dept. of Transportation's intelligent vehicle initiative, and also as a front-end to simulation-based systems.

The ATON tasks carried out at CREATE involved four main areas:

- 1) distributed programming infrastructure for real-time multimedia systems (the

- HPDM and DPE tasks);
- 2) multimedia output for telepresence systems (the CREATE auralizer project);
- 3) navigation and interaction in VE systems (the haptic input project); and
- 4) wireless networking for multimedia applications.

Our previous work (i.e., DRIVE) focused on VE systems architecture, complex world-building tasks, and simple distributed processing for real-time applications. In ATON, we extended and integrated these results with a state-of-the-art communications infrastructure for mobile interfaces, and middle-level software APIs and support libraries for transparent distribution of multimedia processing and I/O functionality.

The ATON project's voluminous documentation is available from the Web site <http://www.create.ucsb.edu/aton>. We will discuss the two main tasks related to distributed systems below as separate projects: HPDM and DPE. We intend to continue these tasks even after the end of the ATON project.

HPDM: High-Performance Distributed Multimedia Support

To support the kind of real-time distributed object software we're building, we need a uniform (LAN- or WAN-oriented) distributed object programming library based on multi-vendor standards such as CORBA. The CREATE *High-Performance Distributed Multimedia* (HPDM) effort built a high-level API for distributed multimedia software that includes the standard CORBA-style features and adds new features such as quality-of-service management and load-balancing.

Any distributed programming library and tool-kit must include at least two components:

- 1) a standardized *external data representation* (XDR), which describes how data is passed over the network in a machine-independent fashion, and
- 2) a standardized format for *remote procedure calls* (RPC) or (in the case of object-oriented 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 spe-

cial *interface definition language* (IDL).

Two other required 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), as well as describing the run-time environment (e.g., special I/O devices) a service uses.

Typical distributed programming frameworks (e.g., CORBA or Java RMI) 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 converters 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 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 CORBA specification standardizes XDR, RMI, and "event channels" (as an optional service that is not provided by all ORBs) 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 (i.e., bounded latency) is a requirement, it may be difficult to characterize the suitability of CORBA RMI or event channels.

Even the emerging real-time CORBA standard focuses on fixed priority scheduling

algorithms, which are unsuitable for applications with non-deterministic and non-static QoS requirements (such as we are building). ATM networks provide a fine-grained model of QoS, but few higher-level libraries provide application-level "hooks" that allow programmers to make simple use of these facilities.

To improve on the weakness of the CORBA standard (no QoS annotation or scheduling control), and the fragility and non-portability of several of the current proposed solutions (CORBA-RT, CORBA-A/V), we decided to design and implement a set of simple extensions to the CORBA IDL language, and to make a set of simple lightweight services (and associated tools) for running our systems.

The HPDM Multimedia IDL

In the first stage of the HPDM project, we designed and implemented an extended *Multimedia IDL* (MIDL) that allows us to incorporate QoS annotations and run-time information into CORBA interface descriptions. The multi-pass MIDL compiler reads standard IDL and generates source code files in the language of choice (C++, Java, or Smalltalk), and takes the extended method annotations and stores them in a database for later use by the run-time system described below.

The MIDL extensions focus on several areas that the CORBA interface definition language (and even the recent real-time and messaging extensions) don't address. To support HPDM applications, there are four kinds of extensions we need to integrate with CORBA IDL definitions:

- quality-of-service (QoS) information (e.g., allowable latency and jitter, dropped packet ratio);
- method performance characteristics (e.g., run-time scaling with data size);
- interface hardware requirements (e.g., needs special HW); and
- method run-time characteristics (e.g., called at the frame rate).

To capture and use this information, we need to maintain several databases or repositories that are derived from MIDL descriptions and auxiliary information. These include:

- IDL interfaces - data structures, function signatures;

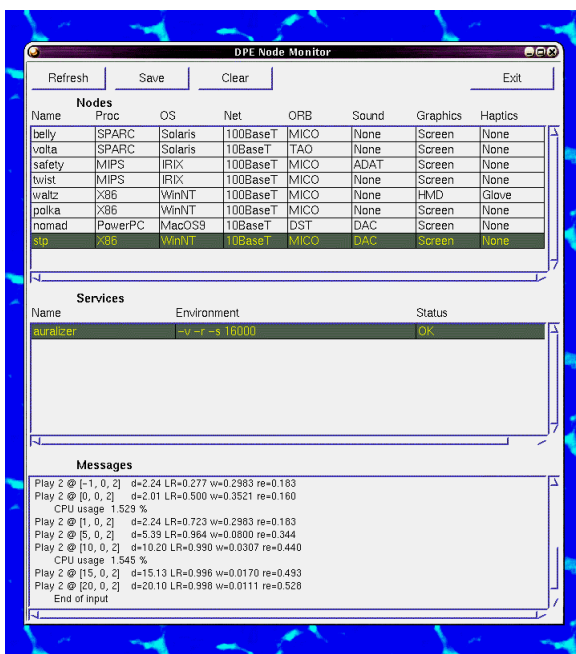
- MIDL annotation - QoS, etc.;
- network configuration (machines, MIPS, network, HW, I/O, etc.); and
- scenarios (examples of distributed application configurations).

We have built a first-generation MIDL compiler that uses a relational database as its interface repository, and have developed several HPDM scenarios for testing the system.

Management of Distributed Processing Environments

- To run HPDM/MIDL programs, we need:
 - A basic remote object infrastructure such as a set of CORBA ORBs (one per machine, not necessarily identical);
 - A naming or trading service for locating object references, possibly based on intimate knowledge of the run-time environment and the interface used;
 - An event publish/subscribe system that allows many-to-many asynchronous message-passing with efficient event filtering and routing; and
 - A run-time task manager for start/stop/monitor/tune operations (see Figure).

Each of these components is the subject of a design and prototyping task as part of the *Distributed Processing Environment (DPE)* project.



The DPE Manager GUI

The DPE is an infrastructure for configuring and managing distributed software systems. It consists of several components that allow the start-up, monitoring, and shut-down of CORBA components on a computer network. The DPE addresses both the issues of fault tolerance and load-balancing, though these concerns are secondary to basic system start-up and shut-down in the initial implementation.

The current DPE manager GUI is shown in the Figure; the top list shows the machines that are participating in the current application(s), the middle list shows the services running on the selected machine, and the lower text view displays the messages from the selected service.

The basic DPE infrastructure consists of a small manager program that runs on every machine in the network. This program implements the IDL *NodeManager* interface, which supports sign-of-life tests (e.g., ping()) and regular heartbeats, basic resource and system activity queries (e.g., available_CPU()), and spawning new services.

Each DPE-managed service implements the IDL *DPE_Service* interface, which includes the sign-of-life and heartbeat calls, and controlled shut-down and restart of services. More advanced DPE implementations will use the run-time information derived from the MIDL to automatically distribute and tune HPDM applications.

As an example, we present the following scenario. Four nodes are to be used to run a distributed VR system. The following services are to be distributed and managed by DPE on four machines:

- waltz (PC/NT)—input tracking (hand/head trackers & glove);
- polka (PC/Linux)—stereo-optic visual rendering;
- safety (SGI/IRIX)—spatial sound rendering; and
- belly (Sun/Solaris)—CORBA name server, database, and sound server.

A node manager program must be started on each machine (probably as a start-up service). The DPE manager is started by hand on the manager machine and reads a system configuration from the database. It sends calls to the appropriate node managers to start their

heartbeats (sending regular activity messages to the DPE manager). The DPE manager then proceeds to send start-up messages to the nodes to spawn the various services.

During execution of the system, the DPE manager monitors the four machines to make certain that the services are still active, and that the load on the machines (in terms of CPU cycles, network I/O, or other dimensions) is within operational limits.

Given the DPE and applications that use it (i.e., that have their interfaces described in MIDL), we can carry out detailed performance tests, and determine which language/hardware/network combinations suit themselves for which kind of multimedia applications (e.g., to answer the question “can we run this configuration of DRIVE over an OC-3 ATM network?”). Using this first-generation HPDM DPE, we intend to address the issues of data streaming and caching, run-time management, and fault tolerance.

It should now be obvious that the HPDM/DPE infrastructure allows one to schedule and maintain the kinds of networks of processing and I/O servers that are needed for large real-time systems in a wide range of applications areas. We have also made use of the best technical features of the standards and tools we use (e.g., CORBA), and augment where necessary with new solutions to the problems we encountered.

Plans for Future Work

We are still realizing the vision of “Next-Generation Networked Multimedia” using the tools we have built for real-time distributed multimedia systems. We are currently planning to continue our work on the HPDM framework, MIDL system and the DPE tool kit. Each of these tasks are the subject of more in-depth project proposals. The concrete projects we are planning address the following topics:

- DPE-managed applications that provide fault-tolerance and load-balancing;
- CORBA applications that provide run-time quality-of-service negotiation;
- MIDL pre-processors for commercial ORBs that allow extended QoS models;
- Wide-area distributed real-time multimedia applications such as telepresence-

- based user interfaces;
- Efficient many-to-many publish/subscribe services for HPDM applications;
- GUI-based tools for configuring, managing, and monitoring distributed (possibly real-time) applications;
- Development and management of distributed applications that combine event-based and streaming multimedia data interchange;
- Distributed user tracking and rendering for large-scale multi-user virtual environments; and
- Scalable surround sound synthesis and rendering.

It is important to remember that most of the technology described above could be used for any large-scale real-time applications, including, but not limited to, the following areas:

- Internet-based media databases and content delivery systems;
- multi-user simulations and games;
- application server farms;
- high-performance multimedia processing for rendering and mixing; and
- fault-tolerance and load-balancing for reliable systems.

Contact Information

Stephen Travis Pope
Email: stp@create.ucsb.edu
Tel: (805) 967-2621

Updated 2001/3/16