HDIMI: Heterogeneous Distributed Multimedia Information Management for QoS-Sensitive C^4I Applications in the Global Infosphere

Wonjun Lee* · Jaideep Srivastava** · James Richardson***

1. INTRODUCTION

The *Heterogeneous Distributed Information Management for the Infosphere (HDIMI)* project is conducting research and development in information management technology to support many operational objectives. The University of Minnesota and Honeywell Technology Center have conducted the HDIMI project under Rome Laboratory (now U.S. Air Force Research Laboratory) sponsorship. The objectives of the *HDIMI* project have their origin in *C^4I for the Warrior* [9]. That document stated the importance of providing individual warriors, whatever their role, with relevant and timely information from the global Infosphere. More recently, DoD's *Joint Vision 2010* [2] has emphasized the key role of information superiority in achieving military success. Information superiority leads to enhanced *battle space awareness* (understanding of the current military situation) and *speed of command* (ability to plan and execute operations to meet objectives and to adapt to changing situations). The rest of the paper is organized as follows: In Section 2, we describe the background and objectives of the HDIMI project. Section 3 illustrates the technical approaches of HDIMI. In Section 4, we present the accomplishments of this project, and concluding remarks and future work description will follow in Section 5.

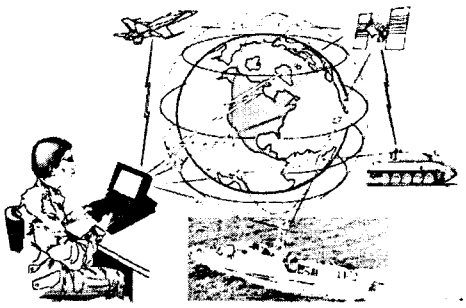


Figure 1. Warriors need relevant and timely information from the global Infosphere

2. PROJECT BACKGROUND AND OBJECTIVES

The HDIMI project is a successor to the

This work is funded by U.S. Air Force contract number F30602-96-C-0130.

*Computer Science Telecommunications, University of Missouri - Kansas City, USA
**Department of Computer Science & Engineering, University of Minnesota, USA
***Honeywell Technology Center, USA

Multimedia Database Management System project conducted in 1993~1996 under Rome Laboratory sponsorship. That project focused on system services and tools to support continuous media (audio, video) in time-critical C^4I applications. Significant accomplishments in that project included:

- A block-based programming model and graphical tool for dynamic construction of complete continuous media applications.
- A multi-resource run-time scheduling component that ensured continued execution of critical applications when system resources are tight, while allowing others to operate with reduced quality of service [15].
- A high-performance multimedia file system.

All of these capabilities were implemented in a Solaris-based system called *Presto*. *Presto* was subsequently extended to a distributed environment under a DARPA-sponsored project, High Performance Network Services [1]. This distributed environment was the starting point for the HDIMI project. We developed extensions to *Presto*'s data-flow oriented, block-based programming model to support operation flows in addition to data flow, and to handle aperiodic flows in addition to periodic flows. These changes were necessary to implement Active View services, and moved the range of applications well beyond the continuous media applications that *Presto* supported.

That is, the HDIMI project has taken many of the concepts embodied in *Presto* and generalized them to address information management requirements implicit in *C^4I for the Warrior* and *Joint Vision 2010*. Specifically, the HDIMI project objectives are to "*investigate, develop, and demonstrate techniques for meeting C^4I system application requirements:*

- A wide variety of information, including conventional data types and continuous media, stored in a collection of heterogeneous data sources in a distributed environment
- A means for each application to defines its 'window on the world' and to specify policies on how closely the window must be kept in synch with the global Infosphere
- The operation and coexistence of QoS-sensitive C^4I applications and other C^4I applications
- Quick and easy prototyping of C^4I applications

…within the framework of an overall layered system architecture."

Significant requirements beyond the capabilities of *Presto* include:

- Support for data types that lack a time dimension, e.g. text, images, and conventional database structures. (*Presto* only supported continuous media.)
- Ability to define a "window on the world", i.e. an application- or user-specific *Active View* of the global Infosphere. (*Presto* supported development of stand-alone continuous media applications. It did not support multiple concurrent accesses to shared databases.)

- More general notions of Quality of Service for these views. (*Presto* supported QoS measures specific to continuous media.)
- Distributed multi-resource management. (*Presto* supported CPU and memory resource management on a single node only.)

3. TECHNICAL APPROACH

The evolution of the *Presto* system to accommodate the new requirements is called *Sonata*. The Sonata reference architecture (Figure 2) has evolved from *Presto* to align with the Joint Task Force Architecture Specification (JTFAS) being developed under DARPA sponsorship [25]. The functional components comprising the architecture fall into five categories:

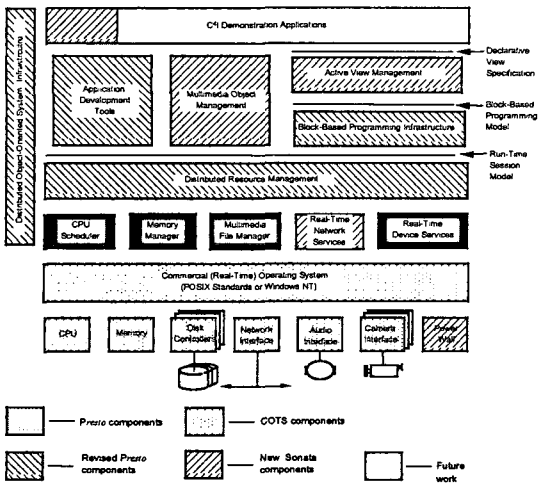


Figure 2. Sonata Reference Architecture

- *COTS components* are commercially available hardware and software products.
- *Presto components* were built during the Multimedia Database Management System project and require no significant revision.
- *Revised Presto components* are those that were initially developed in the Multimedia Database Management System project and need substantial extensions in the HDIMI project.
- *New Sonata components* are being developed from scratch under the HDIMI project.
- *Future work*, such as C^4I applications, can be developed by a follow-on project after successful execution of the proposed HDIMI project.

At the outset of the HDIMI project, we proposed to develop the following six major capabilities. Actual accomplishments are listed in the next section.

- *Active view management*—We will develop Active View capabilities for the multimedia infosphere. This component consists of a declarative view specification model and language, and algorithms that map the views specified in the language to a data-flow-oriented, function-block-based program for execution.
- *Block-based programming infrastructure*—Supporting the Active View management, this component consists of a block-based programming interface and view execution mechanisms. The block-based programming model (and an associated program development tool) was designed and prototyped in the Multimedia Database Management

System project. We will enhance this model with the capabilities of operation flow support and distributed, location-transparent view definition and execution.

- *Application development tools*—We will extend the visual program development tool prototyped in the Multimedia Database Management System project to support Active View specification in addition to application construction. Further, we will develop a user interface tool to facilitate construction of user interfaces for different applications, and a program analysis tool.
- *Multimedia object management*—We will develop capabilities of heterogeneous data type management and content-based query for the multimedia infosphere. We propose to prototype this component using a commercial database management system.
- *Distributed resource management*—This is an extension of the *Presto* work. We will investigate and develop distributed scheduling techniques to support the Active View capability. This component will be built on top of POSIX-compliant commercial operating systems.
- *Application Demonstration*—This indicates various demonstration application software components to be built in the system environment. We will develop a set of software components in the context of a DoD demonstration to illustrate the capabilities of all the system components described above.

4. ACCOMPLISHMENTS

We substantially accomplished the HDIMI project objectives, as summarized below. Subsequent sections of this paper provide further details.

4.1 Active View Management

We defined Active View services, which involve three major concepts: view, change notification, and history. We have implemented the view and change notification concepts in multiple distributed demonstration applications. The service definitions have proved remarkably robust over time. Active View is an object-oriented framework for distributed monitoring and control applications. This is a broad class of applications that includes, for example, military command, control, communications and computing (C^4I) and industrial process control.

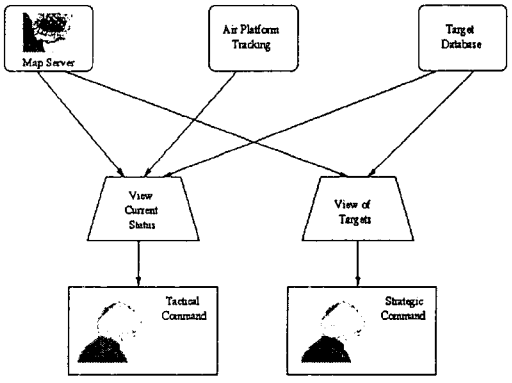


Figure 3. Example Distributed C^4I Application

Figure 3 shows an example distributed C^4I application. In this case, the map server, aircraft

tracking, and target information are dynamic sources of information. Different parts of the organization are interested in different subsets of the aggregate information. For example, the strategic command is interested in target reconnaissance videos, and various views of the targets to help plan missions. The tactical command is interested in the status of current missions, and their feasibility, both in terms of resources, and timeliness. Having up-to-date information is crucial for good decision making in either scenario. As this example illustrates, this framework has some special needs. First, changes in the situation being monitored must be propagated to the end user/application with appropriate timeliness and information quality guarantees. Second, a wide range of data types, including continuous media like audio and video, and others like text, images and records, must be managed. Finally, there is the requirement to use commercial object-oriented technologies as much as possible.

4.2 Block-Based Programming Infrastructure

It has been observed that the current programming paradigm for developing multimedia software needs some improvement [17,26]. *Presto* used a data-flow oriented, block-based programming model and execution environment for continuous media applications. The model is based on a data flow programming approach to facilitate construction of continuous multimedia applications. A multimedia application is visualized as a directed graph, wherein the nodes represent common generic multimedia modules and the edges depict the flow of multimedia streams. The nodes in the graph, called *blocks*, represent operations that modify streams as they flow through them. The operation parameters of a block are specified through parameter ports. The multimedia streams flow through data ports. Application functions are implemented as blocks and applications are "programmed" by interconnecting their functional blocks. Thus, the model enables the plug-and-play programming paradigm, making application programming easy and efficient and supporting reuse of application software. A *program* is an application programming model for describing continuous multimedia applications based on the data flow paradigm. It consists of a set of blocks interconnected through data ports by media flow paths. Figure 4 shows an example of a video-capture-process-display program comprising video camera, motion detection, color filter, and display blocks.

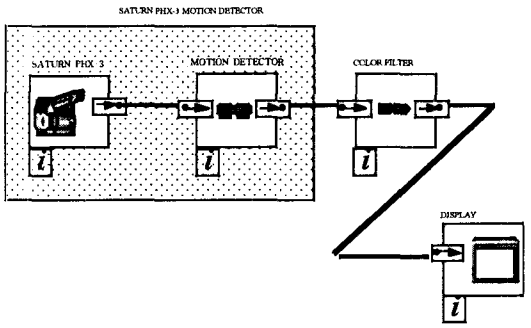


Figure 4. Example Block-Based Program

A block consists of:

- A vector of input ports that accept

incoming data streams

- A vector of output ports that produce outgoing data streams
- A vector of parameter ports that are used to set operating parameters
- A function that produces output streams, consumes input streams, or transforms input streams into output streams,
- A pair of matrices for data rate and QoS translation between input and output ports.

A block is called *basic block* if its function is coded in a language such as C++ or *composite block* if it is formed by assembling and connecting a set of basic blocks. Figure 5 shows example basic program blocks.

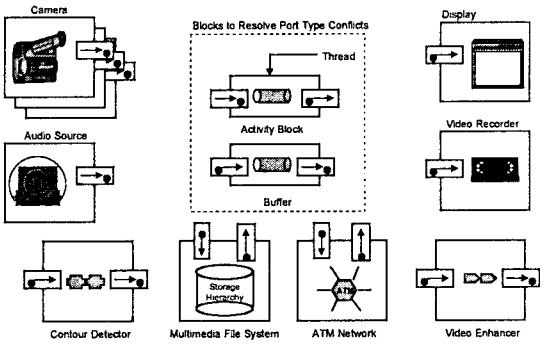


Figure 5. Example Program Blocks

A *port* is an interface of a block that captures interactions with other blocks. As illustrated in Figure 6, it is defined by a data type (JPEG, audio, etc.), a data flow direction (input or output), and a control flow type (push or pull). In push-type control flow, the output port takes the initiative to deliver data to the input port. In pull-type control flow, the input port requests data from the output port. Therefore push output ports and pull input ports are active, whereas pull output ports and push input ports are passive.

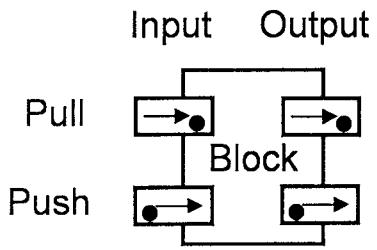


Figure 6. Port Types

4.3 Application Development Tools

Presto includes a Program Development Tool (PDT). Using the PDT, a user can construct a program from a library of basic blocks, composing them by connecting output ports to input ports without regard to push/pull port type compatibility. Such a program is called a *user program*. The Sonata PDT is an evolution of the *Presto* PDT. *Presto* transforms a user program to a *system program*, as illustrated in Figure 7, by locating push/pull incompatibilities and correcting them by inserting special blocks. If a push output port is connected to a pull input port (both ports are active), *Presto* inserts a buffer block between them. If a pull output port is connected to a push input port (both ports are passive), *Presto* inserts an activity block between them. Separation of the user program from its corresponding system program makes the control flow—the push/pull semantics—transparent to the application programmer and

simplifies block-based programming.

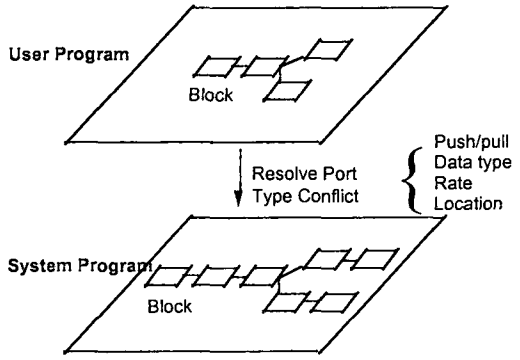


Figure 7. User Program to System Program Translation

The *Presto* block-oriented programming model was inspired by other work on process control applications [22], graphical application development [6,8,10], and commercial simulation tools. However, the previous work did not explicitly address the system integration and timing issues encountered in constructing continuous multimedia applications. *Presto* was unique in providing a software methodology that integrates user-level application development and system-level application execution and in extending the block-oriented programming model with rate and QoS properties to support the temporal constraints of multimedia applications.

We re-implemented *Presto*'s Program Development Tool (PDT) in Java. The previous version depended on a Smalltalk-based "meta-tool" called DoME that, while powerful, required too much specialized knowledge to use. The Java implementation will permit relatively easy porting to other platforms in the future. We developed a User Interface Development Tool (UIDT) to construct application user interfaces in a "visual" manner consistent with Active Views concepts. The UIDT is integrated with the PDT, so that the same application can easily be viewed from either perspective. We have achieved levels of tool/run-time integration and data type support that are significant advances over *Presto*:

- In *Presto*, the PDT was used to define a complete program, from a continuous media source (e.g. camera or video file), through data transformation functions (blocks) to sink (e.g. file or display). The program was then run as a unit on one or more Sun workstations.
- In Sonata, programs are built incrementally. Data sources include ObjectStore class extents *and* continuous media sources; views can be built on top of them, and on top of previously defined views. The CORBA name service holds the set of data sources and views available at any given time. These are visible in the PDT for further view construction. Applications can be built and executed *incrementally*, adding new views (blocks) on top of views that are already active in the run-time environment. These applications run in a distributed environment, linked via CORBA.

Applications developed using PDT and UIDT can be targeted to multiple execution environments. In addition to targeting applications to the Sonata run-time system, the tools can develop applications for the Berkeley Continuous Media Toolkit run-time [20,17]. For brevity, we will

not cover the details of these application development tools in this paper.

4.4 Multimedia Object Management

We evaluated a number of COTS object-oriented and object-relational database management systems to use as a basis for persistent object management and query. We selected Object Design Inc.'s ObjectStore server. We developed tools to facilitate creation of Active Views of arbitrary ObjectStore schemas. We extended *Presto*'s continuous media file system into a Continuous Media Server (CMS) that supports concurrent retrieval of multiple media streams by distributed clients. CMS is integrated with Active Views—The PDT and UIDT can be used to develop applications that access information from both ObjectStore and CMS. We demonstrated that *Presto*'s continuous media file system can perform significantly better than the standard UNIX file system. The Continuous Media Server is described further in subsection *Continuous Media Server*. The COTS DBMS evaluation is reported separately [18]. With AFRL concurrence, we decided not to investigate content-based query of multimedia data. The combination of content-based query and Active View technology is a powerful one for automating intelligence data analysis. For instance, one could define a view that lists enemy tanks in a specified geographic region, given a set of raw images. Computing the view requires executing image analysis algorithms. Our approach had been to integrate existing algorithms in the Active View framework, rather than to innovate in image analysis. However, we believe that the current state of the art of image analysis algorithms is insufficient for a compelling demonstration.

4.5 Distributed Resource Management

Resource management was a major focus of the Multimedia Database Management System project. *Presto* included a component that performed admission control and adaptive multi-resource scheduling based on applications' Quality of Service (QoS) needs. We had planned to extend this capability to a distributed environment in Sonata. However, in concurrence with AFRL, we decided not to pursue this objective for several reasons:

- Applying project resources to other objectives (e.g. a more substantial demonstration) was more valuable.
- With the conversion to a CORBA-based run-time infrastructure, the resource management architecture would have required a redesign.
- The resource management concepts developed in the Multimedia Database Management System project are being extended under a separate project funded under DARPA's Quorum program [7].

Thus, we replaced *Presto*'s custom-built distributed execution environment with one based on CORBA. Specifically, we use Iona's Orbix product. While this involved replacing

major portions of the Sonata code, we believe it provided the best chance of transferring the technology to DARPA programs, such as JFACC, that are based on the JTF architecture. We developed a library of reusable view functions that are building blocks for new applications. These applications can be defined using the application development tools defined below.

4.6 Continuous Media Server

In our Active View System, video processing is handled by a Continuous media (CM) server which has been developed by the University of Minnesota [11]. We discuss this component in the following subsection.

4.6.1 Overview

CM servers have recently been a hot research topic for several reasons. First of all, network speed is increasing, thus, in the near future, services like Video on Demand, Teleconferencing, Distance Learning, etc. are very likely to be popular in everyday life. Given the limitations of current network bandwidth, however, straightforward TCP implementations are not suitable for such bandwidth-sensitive applications. TCP has its own flow control mechanisms, error detection and retransmissions, all of which add extra time as well as network bandwidth overhead to the transmission. This causes unexpected and unpredictable delay and jitter time when transferring CM data, while timing is one of the most critical requirements of CM applications. Most CM applications do not need highly reliable transmission. Losing some frames is less important than having too much delay jitter or losing synchrony between streams. A convincing fact is that a typical TCP connection bandwidth is 2.6 Mbps on a LAN, 580 Kbps on the MAN, and 104 Kbps on the WAN. While for UDP, they are 9 Mbps, 5 Mbps, and 1.2 Mbps respectively on LAN, MAN and WAN. Obviously, TCP is not a good candidate for high bandwidth media streaming.

Given that observation, the natural questions are:

- Is UDP suitable for CM applications?
- How good/bad is it?
- What are the criteria (QoS) for evaluating it?
- If it is bad, how do we reduce the lossy property of UDP while still making use of its higher capability of bandwidth for applications that are speed-sensitive like CM servers?

Secondly, the loss of UDP packets sent over a network is usually caused by buffer overflow. Experiments show that if a sender keeps pushing UDP packets onto the network, even if network bandwidth is good enough to handle it, there still are some lost packets because the consuming time of the receiver is quite large. This applies perfectly to client/server kind of application. For example, with video streaming, the consuming time of a client depends largely on the capability of video cards, which are not always good. This is even worse for audio streams, an 8 kHz audio

stream (e.g. telephone voice) can be played only at 64 Kbps. This delay could cause lots of dropped UDP packets if there is nothing done at the client and/or the server. Moreover, buffer overflow can occur at any of the network switches as well. How do we detect and deal with the fact that the client is good enough to handle data but network congestion limits the stream reliability. Next, the fact that human ears are a lot more sensitive to interrupts in voice than human eyes to interrupts in video frames, raises up another natural question: how do we deal with loss of UDP packets in loss-sensitive stream like audio? Moreover, given limited resources like network bandwidth, I/O time (disk seek, latency time), memory capacity, and CPU time, the capability of a CM server shall obviously be reduced as the number of streams (clients) increased. A best effort strategy is simple, but a preferred policy is to deny a request if the CM server knows that it is not able to handle the request. An appropriate admission control algorithm [15] must be adopted for this purpose. Even if all the above problems have been solved, inter-stream and intra-stream synchronization are questions next to be answered. Lastly, we ported our socket-based CM server with CORBA. We used Orbix version 2.0 from IONA Technologies for the CORBA implementation. CORBA-version CM server replaces all C socket calls with stubs and skeletons generated from a pair of CORBA interface definition language (IDL) specifications. Due to the higher fixed overhead of CORBA such as demultiplexing and memory management, this version shows much lower performance. In the following subsection, we will present a design and implementation of a prototyped QoS-driven CM server.

4.6.2 Design of CM Server

Our CM server system is a typical client/server application. It includes one CM server and multiple CM clients. The CM server has four major components. The Network Manager responds to clients' connection requests. The QoS Manager is responsible for admission control and I/O scheduling. Each Proxy Server communicates with a client, receiving CM stream operation requests and sending CM data by network. Each I/O Manager reads out CM data from disks for a proxy server. The CM client is relatively simple compared with the CM server. The CM client requests stream-operations (such as open, play, pause, and close) to the CM server, and receives data from the server—in some rate (given by client)—as well as displays the retrieved stream on the screen. It has two main modules: Client N/W Controller and CM Player. The client N/W Controller communicates with CM server. It is responsible for forming requests for starting CM streams, changing playback rate and other QoS parameters, and stopping the connection. Once the CM stream is started, this module keeps receiving data and puts them into common buffers. The client CM player periodically gets a logical data unit from the common buffers and plays it on the relevant display device. Our CM server has several versions to support various Internet protocols and environ-

ments such as TCP, UDP, and CORBA.

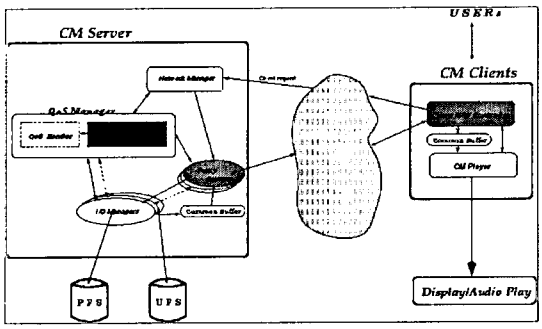


Figure 8. Architecture of Continuous Media Server

4.7 Using Commercial Off-the Shelf (COTS) Products

For many reasons, including development cost and standards in the application domain, we decided to use commercial off-the-shelf products for distributed object services and persistent object management. Choosing the best products for our particular needs was quite important. The COTS products that we picked were Iona's Orbix and Object Design's ObjectStore. The application domain standards mandated the use of CORBA [15]. We chose Orbix in view of its dominant presence in the Unix environment. The selection of an object database was much more difficultsince there are a plethora of products, but no standard. We chose ObjectStore, based on a survey of OODBMSs [18]. The choice of these products strongly affected the design and implementation of our system.

4.7.1 Object Systems Interoperability

We were faced with the problem of handling three different object systems, i.e., the one that comes with the programming language (C++), the distributed object management system (CORBA/Orbix), and the object database system (ObjectStore). There is enough difference in the three approaches and their abilities that we had to consider interoperability issues. For example, CORBA and ObjectStore have different object granularities—we could model an ObjectStore collection as a CORBA object, but the individual objects that comprise such a collection couldn't be easily fit into the CORBA model. The CORBA model defines an object by its interface, while the ObjectStore model is tied to the implementation of an object. This tension both helped and hampered us. It helped us in that the two systems affected different parts of the design, and changes made for one did not affect the other too much. It hampered us by increasing the number of variables to deal with in the overall design and implementation.

4.7.2 Distribution

A related issue was that CORBA and ObjectStore have different models of distribution. This strongly affected our model of distribution. CORBA was mainly useful in making our framework support distributed applications. We discovered that, since our framework imposed certain requirements on the style of code written, making it distributed was relatively easy. The transition to using CORBA was reasonably painless, once we understood the conceptual differences in the object models. The ObjectStore model of distribution affected the details of the data transfer

among communicating distributed objects.

4.7.3 Database Design

ObjectStore's implementation exposed *reference* semantics, which are not naturally modeled by conventional database views. We thus had to make some basic changes to our model to accommodate this.

4.8 Other Issues

Some of the other issues that we considered in designing our framework are:

4.8.1 Choice of an Object-Oriented Language

The choices were C++ and Java. C++ has more mature off-the-shelf products, but Java claims to be the language of choice for distributed objects. We felt that C++ should be the language used, since most of the COTS products for Java weren't mature enough at the time. We used C++ templates extensively to enforce interfaces, without being too dependent on a class hierarchy to provide it. This was of great help when we had to modify our class hierarchy to accommodate Orbix and ObjectStore. Also, the concept of template *traits* helped us hide the differences in accessing persistent and non-persistent objects, and differences in accessing local and remote objects.

4.8.2 Real-Time and QoS

We envisioned the application being used interactively, with the presentation of multiple related data-types at the same time. For example, one window would show a map, another would show a video of the same location, and another would list the resources in that area. In enforcing real-time and QoS requirements [28], we were strongly limited, since we had no good model of the behavior of the COTS products that we used. Due to this, we have not yet addressed these issues in our implementation.

4.9 Demonstration Application

We developed several demonstrations in the course of the HDIMI project. We developed technology-oriented demonstrations of Active Views, the Continuous Media Server, and the application development tools. All of these technologies have been incorporated into a larger demonstration that shows how Sonata technologies apply to air combat planning and monitoring. Section *Demonstration Application* describes this demonstration. Instructions for operating the demonstration software are documented separately [27].

5. CONCLUSIONS & FUTURE WORK

The HDIMI project has focussed primarily on developing broadly-applicable information management technology as opposed to C^4I applications. The technology is sufficiently mature that it should be used and evaluated in the context of application-oriented programs such as DARPA's JFACC, Dynamic Database, and Adaptive Information Control Environment (AICE) programs. To this end, the University of Minnesota and Honeywell Technology Center have

submitted a number of proposals to DARPA. All have been deemed "selectable". We continue to pursue this course of action and solicit AFRL assistance. A second path that can be pursued simultaneously is to extend the capabilities of the existing Sonata technology.

6. ACKNOWLEDGEMENTS

We would like to thank Mr. Mark Foresti and many colleagues at U.S. Air Force Research Laboratory for their many valuable suggestions and sponsorship. The ideas implemented in the HDIMI project have benefited a lot by discussion with our colleagues, Raja Harinath, Difu Su of University of Minnesota, Prof. Duminda Wijekesera of George Mason University, and Dr. Deepak R. Kenchamanna-Hosekote of IBM Almaden Research Center, USA. We would like to thank all of them for their valuable suggestions and many parts of implementation on this work.

7. REFERENCES

[1] Agrawal, M., Kenchammana-Hosekote, D. R., Pavan, A., Bhattacharya, S., and Vaidyanathan, N. *High Performance Network Services for Multimedia-Integrated Distributed Control.* Technical report, Honeywell Technology Center, Minneapolis, MN, July 1996.

[2] Chairman of the Joint Chiefs of Staff, *Joint Vision 2010*, 1998. http://www.dtic.mil/doctrine/jv2010/

[3] Zhigang Chen, See-moong Tan, Roy H. Campbell, and Yongcheng Li, *Real Time Video and Audio in the WWW.*

[4] Fayad, M., and Schmidt, D. Object-Oriented Application Frameworks. *Communications of the ACM*, Vol. 40, No. 10 (October 1997) 32-38.

[5] Gosling, J., Yellin, F., and the Java Team. *The Java Programming Language.* Addison-Wesley, 1996.

[6] Haeberti, P.E. ConMan: A Visual Programming Language for Interactive Graphics, *Computer Graphics*, August 1988.

[7] Huang, J., Jha, R., Heimerdinger, W., Muhammad, M, Lauzac, S., Kannikeswaran, B., Schwan, K., Zhao W., and Bettati, R. RT-ARM: A real-time adaptive resource management system for distributed mission-critical applications. *Workshop on Middleware for Distributed Real-Time Systems*, San Francisco, RTSS-97.

[8] Ingalls, D., et al. Fabrik: A Visual Programming Environment, Object-Oriented Programming: Systems, Languages, and Applications, *Special Issue of ACM SIGPLAN Notices*, Vol. 23, No. 11, November, 1988

[9] C4 Architecture & Integration Division (J6I), J6, The Joint Staff, *C^4I for the Warrior*, June 12, 1993.

[10] Kass, M. CONDOR: Constraint-Based Dataflow, *Computer Graphics.* July, 1992.

[11] Lee, W., Su, D, and Srivastava, J. *QoS-based Evaluation of File Systems and Distributed System Services for Continuous Media Provisioning.* To appear in Information & Software Technology, Elsevier Science, 2000.

[12] Lee, W. and Sabata, B. *Admission Control and QoS Negotiations for Soft-Real Time Applications,* in Proceedings of the IEEE International Conference on Multimedia Computing Systems (ICMCS), Florence, Italy, June 1999.

[13] Lee, W. and Srivastava, J. *An Algebraic QoS-based Resource Management Model for Competitive Multimedia Applications,* To appear in Journal of Multimedia Tools and Applications, Kluwer Academic Publishers, 2000.

[14] Lee, W., Su. D, Srivastava, J., Kenchammana-Hosekote, D.R., and Wijesekera, D., *Experimental Evaluation of PFS Continuous Media File System,* ACM International Conference on Information and knowledge Management (CIKM 97), Nov. 1997.

[15] Object Management Group. *The Common Object Request Broker: Architecture and Specification. Revision 2.0.* July, 1995.

[16] Ousterhout, J. *Tcl and Tk Toolkit.* Addison-Wesley, 1993.

[17] Patel, K. *Introduction to the Continuous Media Toolkit (CMT).* Berkeley Multimedia Research Center, 1995.

[18] Pazandak, P., and Srivastava, J. Evaluating Object DBMSs for Multimedia. *IEEE Multimedia,* 4, 3 (July-September 1997) 3449. Also submitted to AFRL as a COTS technology evaluation report under the HDIMI contract.

[19] Schmidt, D., and Fayad, M. Lessons Learned: Building Reusable OO Frameworks for Distributed Software. *Communications of the ACM*, Vol. 40, No. 10 (October 1997) 8587.

[20] Smith, B.C., *Implementation Techniques for Continuous Media Systems and Applications,* PhD thesis, University of California, Berkeley, Computer Science Department, 1994.

[21] Steinmetz, R. Human Perception of Jitter and Media Synchronization, *IEEE Journal on Selected Areas in Communication*, Vol. 14, No. 1, pp. 61-72, 1996.

[22] Stewart, D.B.. *Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects*, Technical Report CMU-RI-TR-93, Advanced Manipulators Laboratory, The Robotics Institute, and Department of Electrical and Computer Engineering, Carnegie Mellon University. July, 1993.

[23] D.B. Stewart, R.A. Volpe, and P.K. Khosla. *Design of Dynamically Reconfigurable Real-Time Software using Port-based Objects.* Technical Report CMR-RI-TR-93-11, Department of ECE, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, July 1993.

[24] David Tennenhouse, Joel Adam, David Carver, Henry Houh, Michael Ismert, Christopher Lindblad, William Stasior, David Wetherall, David Bacher, and Theresa Chang. The ViewStation: A Software-Intensive Approach to Media Processing and Distribution. *Multimedia Systems* Vol 3, pp. 104-115, 1995.

[25] Teknowledge Federal Systems, *Joint Task Force Architecture Specification*, April 13, 1994. Updated information available at http://itfweb3.nosc.mil/.

[26] Thompson, J. and Gottlieb. *Macromedia Director Developers Guide to Lingo*, 1995.

[27] *Sonata User and Administration Manual*, submitted to AFRL as an HDIMI project deliverable, 1998.

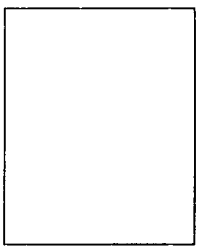
[28] Wijesekera, D, and Srivastava, J. Quality of Service (QoS) Metrics for Continuous Media, *Multimedia Tools and Applications*, Vol 3, No. 1, pp. 127-166, September. 1996.

Wonjun Lee

Dr. Lee is Assistant Professor of the School of Computer Science Telecommunications at the University of Missouri - Kansas City, USA. He received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, Korea in 1989 and 1991, respectively. He also received the M.S. in computer science from the University of Maryland, College Park, USA in 1996 and the Ph.D. in computer science and engineering from the University of Minnesota, Minneapolis, USA, in 1999. His research interests include networked multimedia computing, distributed systems, real-time systems, databases, and Internet technology.

Jaideep Srivastava

Dr. Srivastava is Professor of the Department of Computer Science and Engineering at the University of Minnesota, Minneapolis, USA. He received the Ph.D. and M.S. degrees in computer science from the University of California, Berkeley, USA in 1988 and 1985, respectively. He received the B.S. in computer science from Indian Institute of Technology (IIT), Kanpur, India in 1983. His major research interests include Distributed Systems, Multi-Media Computing, Data Mining, Web Mining, and Database Integration.



James Richardson

Dr. Richardson is Technical Staff of Honeywell Technology Center, Minneapolis, USA. He holds a Ph.D. in Computer Science from Stanford University. His research areas include Database Systems, Real-Time Computing, and Multimedia Systems.