네트워크 멀티미디어 시스템 구현 이슈 및 QoS 협상 프로토콜 모델

이 원 준†

요 약

본 논문에서는 분산 멀티미디어 응용을 위한 객체 지향형 프레임워크 방식으로 설계된 네트워크 멀티미디어 관리 시스템 개발에 관한 설계 이슈 및 프로토타입 개발 경험을 기술하고, 특히 멀티미디어 관리 시스템의 일부로서 개발한 비디오 서버 상에 적용 가능한 통합형 QoS-자원 협상 프로토콜의 주요 특징에 관하여 설명한다. 구현된 멀티미디어 프레임 워크 상에서 효율적인 멀티미디어 스트리밍을 지원하기 위하여 새로 제안된 QoS 협상 정책을 실제 서버에 적용하는데 있어서 고려해야 할 중요 이슈에 대해서도 분석하였다.

Design Issues and QoS Negotiation Protocol Model for Networked Multimedia Systems

Wonjun Lee†

ABSTRACT

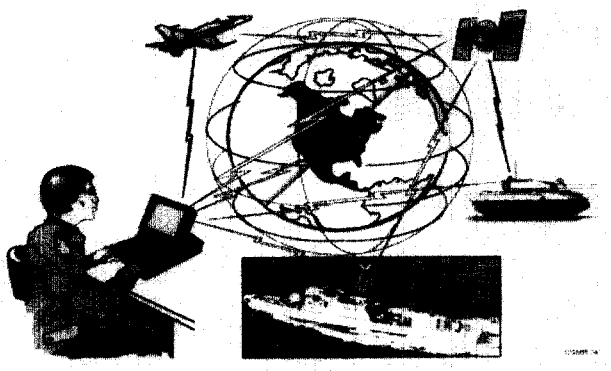
This paper describes our experiences with the design and implementation of a networked multimedia information management system in an object-oriented framework for distributed multimedia applications, and an integrated QoS-resource negotiation protocol which has been applied to a video server in our networked multimedia infrastructure. The salient features of our framework to support efficient multimedia streaming are explained. Next the paper explores the challenges faced in integrating the proposed QoS negotiation policy into the framework.

키워드 : 네트워크 멀티미디어(Networked Multimedia), QoS 협상, 분산 멀티미디어 시스템(Distributed Multimedia Systems), 비디오 서버(Video Server)

1. Introduction

The *Heterogeneous Distributed Information Management for the Infosphere* (*HDIMI*) project has been conducting research and development in information management technology to support many operational objectives. 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). In this paper, we will aim at developing an integrated QoS-resource negotiation protocols which could be applied to the video server system developed as a major portion of the whole HDIMI system. 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 the project. In the following section, we will focus on the design objectives of a continuous media server system and a QoS adaptation protocol for the server. Concluding remarks and future work description will follow in Section 6.

※ This work was supported by Korea Research Foundation Grant (KRF-2001-003-E00219).
† 종신회원 : 고려대학교 컴퓨터학과 교수
논문접수 : 2002년 7월 31일, 심사완료 : 2002년 8월 19일



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

2. Design Objectives

The HDIMI 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. This distributed environment was the starting point for the HDIMI. We developed extensions to existing 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. Specifically, the HDIMI objectives are to "*investigate, develop, and demonstrate techniques for meeting C^4I system application requirements* :

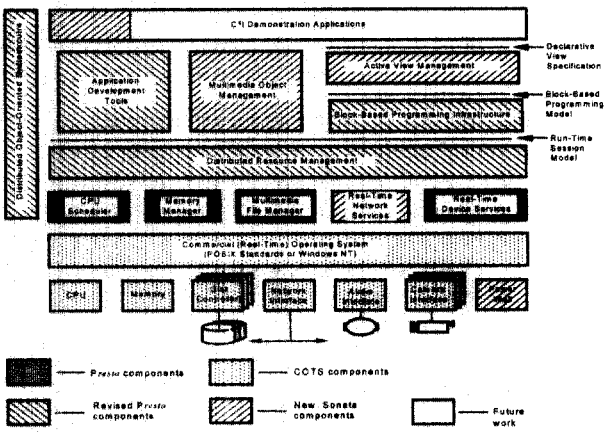
- A wide variety of information, including conventional data types and continuous media, stored in a collection of heterogeneous data sources in a distributed environment
- A means for each application to defines its 'window on the world' and to specify policies on how closely the window must be kept 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 include :

- Support for data types that lack a time dimension, e.g. text, images, and conventional database structures.
- Ability to define a "window on the world", i.e. an application-or user-specific *Active View* of the global Infosphere.
- More general notions of Quality of Service for these views.
- Distributed multi-resource management.

3. Technical Approach

The functional components comprising the architecture fall into five categories :



(Figure 2) HDIMI Architecture

- *COTS (Commercial Off-The Shelf) components.*
- *Presto components* [1].
- *Revised Presto components.*
- *New Sonata components.*
- *Future work* such as C^4I applications.

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 specificat-

ion model and language, and algorithms that map the views specified in the language to a data-flow-oriented, function-block-based program for execution.

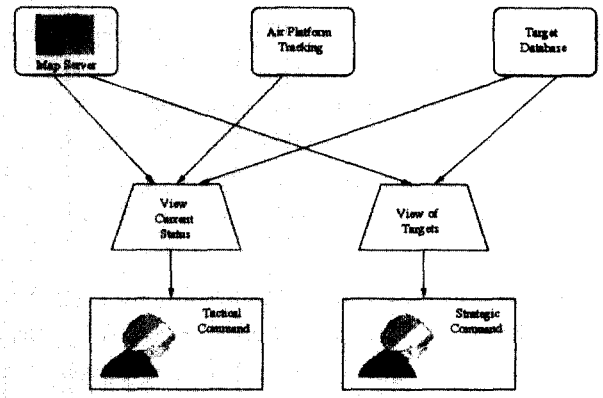
- *Block-based programming infrastructure* : Supporting the Active View management, this component consists of a block-based programming interface and view execution mechanisms. The block-based programming model (and an associated program development tool) was designed and prototyped in the Multimedia 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.
- *Distributed resource management* : 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. Architecture

We substantially accomplished the HDIMI 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.



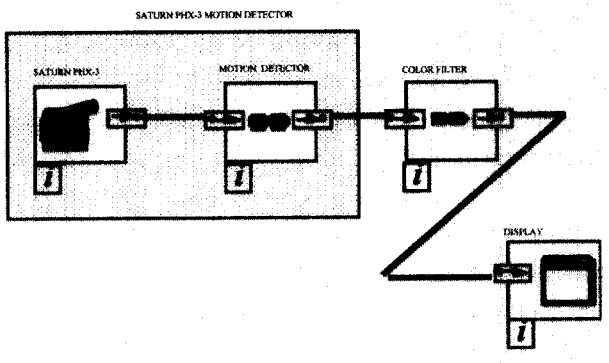
(Figure 3) Example Distributed C4I Application

(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.

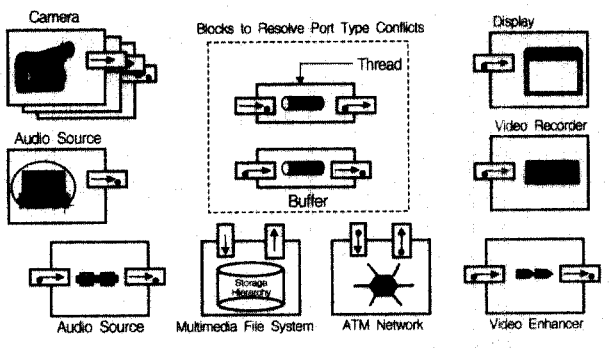
4.2 Block-Based Programming Infrastructure

It has been observed that the current programming paradigm for developing multimedia software needs some improvement [17, 26]. HDIMI 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 5) shows example basic program blocks.



(Figure 4) Example Block-Based Program



(Figure 5) Example Block-Based Program

4.3 Application Development Tools

HDIMI 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*. 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. HDIMI 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. Applications developed using PDT 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].

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 Continuous Media Server is described further in subsection *Continuous Media Server*. 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.

4.5 Distributed Resource Management

Presto included a component that performed admission control and adaptive multi-resource scheduling based on applications' Quality of Service (QoS) needs. 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.

4.6 Continuous Media Server

In our Active View System, video processing is handled by a Continuous media (CM) server which has been prototyped. 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. 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?, etc.

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 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 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.

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 environments such as TCP, UDP, and CORBA.

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.

5. QoS Negotiation Protocols

Much of the work in networked continuous media server systems [14, 20] has focused on either conservative approaches based on bandwidth peaks or statistical methods that model arrival rates and stream bandwidths with probability distributions and determine a satisfactory level of performance by the disk and network subsystems as a system. However, these schemes lack in their theoretical basis to maximize system utility, and they consider a single QoS dimension for adaptation, degradation, negotiation, and renegotiation. We here propose a novel QoS negotiation protocol which is a mixture of greedy and non-greedy strategy. Gre-

edy approach is the most naive approach to admission control and resource adaptation where applications are admitted as long as there are available resources. On application arrival, the policy admits the application if the resources available is greater than the resources requested. On application departure, the policy just adds the released resources from the application to the resources available. A purely predictive strategy is the other extreme of the greedy strategy where one continuously trades off the expected benefit in the future with the benefit from the current application. Non-greedy strategy is more hard to implement because it is not so easy to make an advanced estimate for the future traffic. A desirable compromise is to have a mixture of these two strategies. Our key idea is to assign a portion of the resources as the *reserves* and when the applications start to dip into the reserves, the non-greedy strategy is invoked. As a result, we will ensure that most of the time the admission control is simple using the greedy strategy, but when there is a resource crunch some resources are kept aside for the future important applications.

We specify the resource reserves as *threshold* on the resource utilization. A resource is congested if the resource is opening in its reserve. The admission control protocol is invoked when an application arrives and requests for resources. In case a new application arrives with a certain request for a resource and the resource is not congested, we apply the same policy of greedy strategy. When the resource is congested, the admission control protocol changes to a more conservative policy where smaller and smaller amount of the resources are allocated to the applications. The application agents compute how best to use the resources for maximizing the application utility after the admission control process allocates a certain amount of resources to the application. The application agents tradeoff between the different QoS dimensions of the application and compute the QoS assignment for the application. The QoS negotiation and admission control protocol holds the key to how the current application utility is traded off with the future benefit that the resource can generate for potentially more important applications. There could be several diverse heuristic that we should study in further detail. When applications depart, the application agents return the resources back to the resource pool. These resources are reclaimed by the system and re-distributed to the applications that were admitted with degraded quality. Though there could be various strategies for the redistribution, i.e., resource negotiation, we would propose a strategy which is based on economic Pareto optimality [12] such that the total utility to the applications is optimized. Keeping a portion of the resources in the reserves while there are applications that can use it is inefficient in terms of resource utilization. To improve this performance, we add an additional mechanism that reduces the amount of resources in the reserves if there the resource remains under utilized for a long time period.

To summarize, we show the theoretical background of the integrated QoS and resource negotiation/renegotiation protocols below. Given a QoS vector $\vec{q}_j$, for an application j with the dimension of QoS parameters being l, the object function, R, for resource allocation is given as :

$$R_j(\vec{q}_j), \quad \vec{q}_j = (q_{1j}, \cdots, q_{lj}), \quad j \in [1 \cdots n]$$

If we assume *Rtotal* as the total available resources given, the following equations denote the resource constraints and demand for the application j. Here, x denotes the amount of resource(s) assigned to application j.

$$\sum_{j=1}^{n} R_j(\vec{q}_j) = \sum_{j=1}^{n} R_j((q_{1j}, \cdots, q_{lj})) = \sum_{j=1}^{n} x_j = R_{total}$$

Henceforth, the resource reclamation protocol with regard to QoS among the competing multimedia applications would be modeled using the following two utility functions, which could be solved by constrained non-linear optimization problem theories such that the total utility should be maximized under the constraints given. The former equation can be applied in case of single resource environments and the latter one for multiple resource environments. Here, α_j denotes a relative priority of application j.

$$B_{total} = \sum_{j=1}^{n} \alpha_j \times B_j(x_j)$$

$$B_{total} = \sum_{j=1}^{n} \alpha_j \times B_j(\sum_{j=1}^{n} \alpha_{ij} x_{ij})$$

6. Conclusions & Future Work

The HDIMI 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. We will continue to pursue this course of action and solicit AFRL assistance. A second path that can be pursued simultaneously is to extend the capabi-

lities of the existing QoS provisioning technology in continuous media server systems. Aslo, to verify the practical performance of the proposed scheme, we should be able to conduct wide simulation studies by taking a video distribution application or a mobile resource reservation protocol into account, in terms of diverse QoS metrics quantitatively measured.

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*, 2001. 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, pp.32-38, October, 1997.

[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, 1993.

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

[11] 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.

[12] Cheng, M., The WALRAS algorithm : A Convergent Distributed Implementation of General Equilibrium Outcomes, *Computational Economics, 12 : 1-24*, 1998.

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

[14] Shenoy, P., Goyal, P., et al., Symphony : An Integrated Multimedia File System, *In Proceedings of ACM MMCN '98*, San Jose, CA, 1998.

[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.

[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.

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

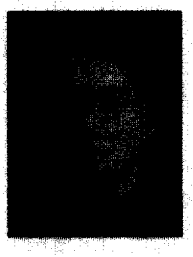
[21] Steinmetz, R. Human Perception of Jitter and Media Synchronization, *IEEE Journal on Selected Areas in Communication*, Vol.14, No.1, pp.61-72, 1996.

[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] Stewart, D. B., 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*, 1994.



이 원 준

e-mail : wlee@korea.ac.kr

1989년 서울대학교 공과대학 컴퓨터공학과 졸업(학사)

1991년 서울대학교 공과대학 컴퓨터공학과 졸업(석사)

1998년 Stanford Research Institute(SRI), Student Research Associate

1999년 University of Minnesota(Ph.D. in Computer Science & Engineering)

2002년 현재 고려대학교 정보통신대학 컴퓨터학과 조교수

관심분야 : 이동 통신, 멀티미디어 시스템 및 네트워크, 분산 시스템