# CReMeS: A CORBA Compliant Reflective Memory based Real-time Communication Service

Sun-Tae Chung* *Regular Member*

## ABSTRACT

We present CReMeS, a CORBA-compliant design and implementation of a new real-time communication service. It provides for efficient, predictable, and scalable communication between information producers and consumers. The CReMeS architecture is based on MidART's Real-Time Channel-based Reflective Memory (RT-CRM) abstraction. This architecture supports the separation of QoS specification between producer and consumer of data and employs a user-level scheduling scheme for communicating real-time tasks. These help us achieve end-to-end predictability and allows our service to scale. The CReMeS architecture provides a CORBA interface to applications and demands no changes to the ORB layer and the language mapping layer. Thus, it can run on non real-time Off-The-Shelf ORBs and enables applications on these ORBs to have scalable and end-to-end predictable asynchronous communication facility. In addition, an application designer can select whether to use an out-of-band channel or the ORB GIOP/IIOP for data communication. This permits a trade-off between performance, predictability and reliability. Experimental results demonstrate that our architecture can achieve better performance and predictability than a real-time implementation of the CORBA Event Service when the out-of-band channel is employed for data communication; it delivers better predictability with comparable performance when the ORB GIOP/IIOP is used.

**Keyword**: CORBA, Reflective Memory, CORBAservices, ORB, Real-time Systems

## I. 서 론

The need to reduce the time and effort required to deploy and maintain distributed real-time applications has necessitated a move towards using off-the-shelf (OTS) distributed object computing middleware based on open standards. OTS middleware typically offer reliable distributed system level support that allows the construction of applications out of reusable software components. The use of these types of components not only reduces the complexity of building the application, but also allows an application designer to concentrate on solving application domain specific problems.

The Common Object Request Broker Architecture(CORBA)[10] specified by Object Management Group (OMG) is beginning to gain acceptance as such a standard middleware platform specification for use even in distributed real-time applications. Some of the problems one faces when using CORBA as part of a distributed real-time architecture arise from its communication primitives being based on method invocations and from its lack of support for expressing and handling timing constraints [19,20,23]. To provide support for QoS requirements of real-time applications, especially those that need communication beyond method invocations, a *Real-Time Event Service* [7,17] has been designed by extending CORBA's *event channel* abstraction [11].

The CORBA Event Service model was originally intended for event dispatching and thus it is only appropriate for applications where asynchronous exchange of small size data message is all that is required. Also, since all asynchron-

* School of Electronic Engineering, Soongsil University (cst@syscon.soongsil.ac.kr)
논문번호 : 00255-0704, 접수일자 : 2000년 7월 4일

ous communication traffic is supposed to go through the event channel, the event channel can become a bottleneck as the number of producers/ consumers using a particular event channel or the size of messages passing through the event channel increases. However, if this bottleneck is avoided by using multiple event channels, the real-time scheduler associated with one event channel may not be aware of messages being exchanged by other event channels and their resource requirements and hence admission control becomes a problem. Therefore, the RT Event Service alone may have difficulty in providing satisfactory end-to-end QoS. Furthermo- re, for real-time applications requiring the trans- port of large data sizes like audio/video, a more scalable and predictable asynchronous communication service is desirable.

In this paper, we propose a new service archit- ecture, called CReMeS (A CORBA Compliant Reflective Memory based Real-Time Communicat- ion Service). CReMeS is based on the concept of Real-Time Channel-based Reflective Memory (RT-CRM) introduced in MidART [6]. It provides for efficient, predictable, scalable, and flexible communication between information suppliers and consumers:

- CReMeS' efficiency derives from the use the Real-Time Channel-based Reflective Memory (RT-CRM) abstraction [21] for communication between nodes.
- CReMeS provides for end-to-end predictability by utilizing MidART's admission control at both the supplier and consumer nodes, and the user-level scheduling scheme for communicating distributed real-time tasks [22].
- CReMeS is scalable since the scheduler in one node handles the communication traffic relating only to that node, as opposed to the scheduler in the Real-Time Event Service which handles all the traffic connecting to the event channel.
- CReMeS' flexibility arises from the fact that application designers can
  - separately specify the QoS requirements for suppliers and consumers, thus allowing different timing properties to be met independently.
  - use a CORBA interface to applications, so no changes to the ORB layer and the language mapping layer are required. Thus, it can run on non real-time OTS(Off-The-Shelf) ORBs and enables applications on these ORBs to have scalable and end-to-end predictable asynchronous communication facility.
  - use an out-of-band channel or the ORB GIOP/IIOP for data communication. This permits a trade-off between performance and reliability.

There are two ways of incorporating new functionality into CORBA. The first consists of integrating the functionality into the ORB core and/or the language mapping layer. But this method loses portability (because the implementat- ion is ORB-dependent) and interoperability (becau- se both clients and servers have to use the same ORB implementation). The second is a Service approach [4] which does not require modification of the ORB core and language mapping layer. In this service approach, the module implementing the new functionality is defined by only an IDL interface and thus is independent of the ORB and language mapping. The latter, service-based, approach is CORBA compliant and hence is desirable.

While a CORBA compliant adaptation of MidART's RT-CRM is desirable - since it is portable and inter-operable, and thus can run on any ORBs - it is not trivial to achieve. One of the major challenges comes from the need to achieve object location transparency while still keeping the efficient shared memory based mechanism that is optimal for intra-node commun- ication. Other challenges include the design of interfaces and adaptation of communication paths of MidART while not losing the performance advantages of MidART.

Keeping the goals of achieving performance, predictability, and scalability in mind, we have designed and implemented a prototype of the CReMeS architecture using OmniORB2 version

3.0.0 [16], an open source ORB, on Windows NT 4.0. Experimental results demonstrate that our architecture can achieve better performance (as measured by latency), predictability (as measured by jitter and the ability to provide QoS guarantees) and scalability (measured in terms of the number of clients or size of messages handled) than a real-time implementation of the CORBA Event Service when an out-of-band channel is employed. It displays superior predictability and scalability while delivering comparable performance when the standard two-way synchronous communication of the ORB GIOP/IIOP is used for data streams. This makes our solution adapt to the needed reliability and performance requirements for building performance sensitive real-time applications in CORBA environments.

This paper is organized as follows: Section 2 provides a brief description of CORBA, and a critique of its real-time capability. In Section 3, we present a description of RT-CRM in the context of MidART. The design and implementation details of CReMeS is presented in Section 4. The experimental performance results of CReMeS implementation are described in Section 5. Finally, we conclude the paper and present a survey of the related work in Section 6.

## II CORBA in Real-time Environments

### 2.1 Overview of CORBA

CORBA, a middleware standard specified by OMG (Object Management Group) [10], defines how objects distributed across heterogeneous distributed environments can be described and can interact with each other. The main component of CORBA is the Object Request Broker (ORB), basically a software bus respon- sible for locating objects and delivering clients' requests to server objects. An ORB provides object location and object implementation transparency in addition to communication infrastructure. CORBA Services, another component of CORBA, are general-purpose high level distributed services on top of ORB to provide convenient functionality to applications. The interface to CORBA Services is defined by OMG. Adopted OMG Object Services, collectively called CORBA services (COS) include the naming service - which allo- ws clients to find objects based on names, and the event service - which supports asynchro- nous events (data) delivery through appropriate event channel implementations. In order to support a general ORB interoperability, CORBA specifies GIOP (General Inter-ORB Protocol) and IIOP (Internet Inter-ORB Protocol) which is the implementation of GIOP over TCP/IP. "Requ- est/Response" type of operation invocations supported by CORBA are usually implemented by a synchronous two-way communication model where clients synchronously wait for response from a server once they invoke operations on server objects. While this type of synchronous two-way communication model is useful because of the similarity of semantics to local object method invocation, synchronous two-way communication can be too costly for certain applications. For example, in industrial control networks and multimedia applications where timely delivery of message and continuous media data delivery are necessary, time taken by a client to wait for server response may cause unacceptable delays. For these reasons, CORBA's Event Service was defined to provide asynchronous communication facility for CORBA applications by decoupling between suppliers of (event) data and consumers of (event) data. CORBA's Event Service can provide anonymous, and one-to-many or many-to-many communication among suppliers and consumers and make connection configuration possible at run-time. Also, recent- ly CORBA adopted Messaging Specification [12], where type-safe Asynchronous Method Invocation (AMI) model is introduced. The CORBA AMI model alleviates some of the problems with the CORBA's standard synchronous communication model, but it is confined to one-to-one communicatlon, and not flexible enough to be configured at run-time. Thus, whereas AMI provides one-to-one two-way asynchronous communication, Event Service provides one-way asynchronous communication and allows group communication. Even though AMI invocations can specify QoS requirements, it does not provide anonymous asynchronous communication and group communication.

### 2.2 Use of CORBA in Real-Time Environments

High performance distributed real-time applications have QoS requirements such as high performance (throughput, low latency), low jitter, and scalability. The successful deployment of CORBA in such applications heavily depends upon the ability of ORB to provide the necessary QoS to applications. But there are both specification and performance problems to consider.

Current CORBA specifications lack several real-time features such as the standard interfaces through which real-time applications can specify their QoS requirements to the ORB, and configure and control ORB resources appropriately for the achievement of desired real-time QoS. Some of these deficiencies have been alleviated with the recent introduction of CORBA Messaging QoS policy specification [12] and Real-Time CORBA specification [13]. But, these specifications are aimed at soft real-time applications. Also, there is still no standard way for clients to indicate timing requirements like latency and jitter of their requests.

Furthermore, compared to low-level programming approaches, current ORB implementations incur significant run-time overheads affecting both throughput and latency, and additional unpredictability [19]. The overheads arise inevitably from the mechanisms which are devised for supporting CORBA features such as object location transparency, object implementation transparency, object method invocation semantics, etc. These mechanisms necessitate presentation layer conversion (Marshalling/De-Marsha- lling), internal message buffering, data copying, demultiplexing, and intra-ORB virtual function calls.

These weaknesses have been studied extensively[19,20,23] and have lead to the development of real-time ORB support. TAO [19] is a high performance ORB (TAO) whose key components include a real-time I/O system, real -time inter-ORB protocol engine, real-time object adapter, real-time scheduling and dispatc- hing mechanism, an optimized IDL compiler which generates efficient and predictable stubs and skeletons. An alternative approach of Wo- lfe et.al. [23] involves extending the current ORB (as opposed to designing a new Real- Time ORB like TAO) to deal with real-time needs. To this end, they have developed a Real-Time manager and object services such as global time service, real-time event service, global priority service, and real-time scheduling to equip current ORB with real-time capability. It should be noted that even these RT-ORBs cannot get rid of all of unpredictability because they don't have complete control of the resources that they need.

An alternative to above extensions to the ORB is to build a higher level CORBA service to provide required QoS to applications. One such service, is the RT Event Service [7,17] developed to extend COS Event Service into real-time application domain. RT Event Service extends CORBA Event Service by supporting periodic rate-based event processing and effic- ient event filtering and correlation. Even though the Real-Time Event Service provides the abo- ve additional features and good performance under light load conditions, it is not scalable nor predictable under heavy load conditions. These drawbacks of the Event Service are mainly due to fundamental architectural coup- ling of each event channel with all of its asso- ciated suppliers and consumers. Our proposed CReMeS addresses these shortcomings.

## Ⅲ. Real-Time Channel-based Reflective Memory (RT-CRM)

Since CReMeS is based on MidART's Real-Time Channel-based Reflective Memory (RT-CRM), in this section, we discuss the relevant aspects of RT-CRM. Although RT-CRM provides for predictable and scalable asynchronous communication infrastructure for applications, it is not a standard middleware. The desire to make its service widely available motivated us to develop CReMeS by adapting RT-CRM for CORBA environments.

RT-CRM was proposed as a service in the MidART middleware. Figure 1 shows the high level architecture of RT-CRM.

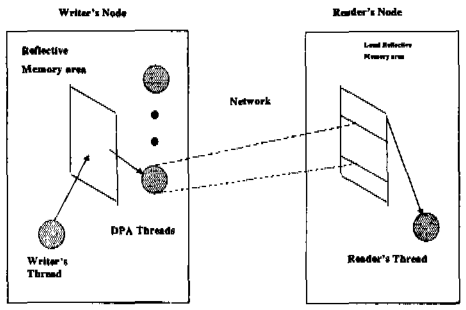


Fig. 1 RT-CRM High Level Architecture

RT-CRM is a software-based reflective memory - it provides data reflection with guaranteed timeliness. Data reflection is defined as the memory-to-memory data transfer among application host memories in a networked environment. Data reflection is accomplished by a Data Push Agent (DPA) residing on the writer's node and sharing the writer's memory area. This agent represents the reader's QoS and data reflection requirements. A virtual channel is established between the agent and the reader's memory area, through which the writer's data is actively transmitted and written into the reader's local memory area by a Data Receive Agent (DRA). RT-CRM provides efficient and flexible asynchronous communication. The run-time communication model of a channel-based reflective memory is defined by the data sending and data reception semantics. Data sending can be either Push-on-Write or Periodic Push, while data reception can be either blocking or non-blocking. Since DPA and DRA are separate threads of control from the application threads, the data sending and data reception semantics are entirely definable and parameterized by the subscriber. The combination of the data push and reception semantics lead to several general models of communications commonly found in distributed real-time applications. For example, Table 1 lists two possible models capable of supporting the requirements of industrial control applications. The two models are: PWB (Push-on-Write, Blocking) and PPN (Periodic Push, Non-blocking).

Table 1. PW=Push-on-Write, PP=PeriodPush, B=Blocking, N=Non-blocking, G=Deadline Guaranteed, NG=No Guarantee

| Modes | Data Types | Deadline | Application |
|---|---|---|---|
| PWB | Sporadic | G | Command issuing |
| PPN | Periodic | G | Trend graph |
| PWB | Sporadic | G | Plant data |
| PPN | Periodic | G | Video/Audio |
| PWB | Sporadic | NG | Background |

## Ⅳ. CReMeS Architecture for CORBA environments

CReMeS implements MidART's RT-CRM as a service for the CORBA environment. In this section, we describe the principal design issues and our implementation experience of CReMeS.

### 4.1 Design considerations and options for CReMeS

Our design goals can be summarized as following:

1. Compatibility with MidART's current usage.
2. Avoiding any performance loss compared to using MidART without CORBA.
3. Providing the application with CORBA development semantics.

We elaborate on how we achieve these design goals in CReMeS below. CReMeS retains compatibility with MidART's original API and their semantics. In MidART, the interface to the RT-CRM service is provided through a library of API similar to memory read-write syntax. In

CReMeS, the interface to the RT-CRM service is constructed as a CORBA object which is defined by IDL and exposed to applications. This CORBA object is called the RT-CRM Interface Object (RIO). Inside the RIO servant (i.e., the entity implementing the CORBA object [10]), a mapping to the reflective memory is provided. RIO methods have the same invocation semantics as MidART's original API. Thus by invoking methods of RIO, an application can set up and use a RT-CRM service. The RIO interface allows applications to specify real-time requirements in terms of deadlines, reader periods, writer periods, and guarantee/non-guarantee mode, and to choose Push-on-Write or Periodic Push mode of commu- nication. This interface is intuitive and much simpler to use than other CORBA communication services such as the RT Event Service. The procedures to use a RT-CRM se- rvice in CORBA environments is similar to those in MidART explained in [25]. The only exception is that an application (writer or reader) now needs to obtain the object referen- ce to RIO before the applica- tion can actually invoke any of the methods.

As for the second goal, there are many design considerations, but two important ones are described here.

- **Location of RIO.**

The location of RIO affects the performance of the communication service. There are two options for the location. One is to locate RIO in an address space different from that of the application, and the other is to locate the object in the same address space as that of the application. If RIO is located in an address space different from he application which manages the reflective memory, as shown in Figure 2, one natural choice of path for data stream transfer between an application (writer or reader) and the associated reflective memory is the ORB GIOP/IIOP path. In this design choice, even asynchronous one-way data transfer needs to pass through the ORB GIOP/ IIOP path twice - once at the writer's end and again at the reader's end - before reaching its final destination. This not only incurs overhead but also increases the jitter. From our initial experimental results obtained with this design choice, the potential for poor performance (increased latency and high jitter) was obvious, especially when transmitting data of large size. In order to achieve performance similar to that of MidART, we need to adopt the shared memory mechanism employed in MidART for data stream transfer between the writer/reader and the reflective memory that it is attached to. In MidART, through the shared memory mechanism, a writer (or reader) can ask for data to be copied directly into (from) the reflective memory in the same node without additional overheads (such as going through IP loopback or additional memory copies). Howev- er, if the servant is located in an address space different from the application, it is difficult to make use of the shared memory mechanism, especially if we want to keep the same interface semantics as MidART's library API.

Thus, the best choice for the location of the RIO is the application's address space (Figure 3): inside the servant of the RIO, a shared memory mechanism for data stream transfer between an application and reflective memory is provided. This data transfer path does not incur much of

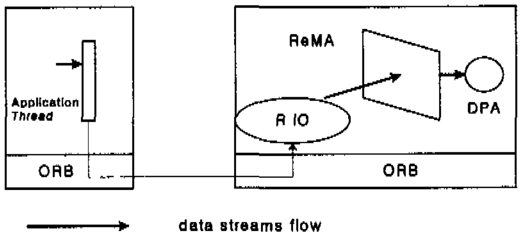


Fig. 2 RIO outside Application

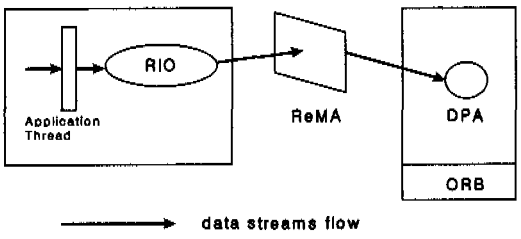


Fig. 3 RIO inside Application space

the ORB GIOP/IIOP overheads since in most of currently existing OTS ORBs (including omniORB2 [16]), object invocation on an object in the same address space (called "collocated object") is optimized to be a virtual function call.

- **Communication path for data reflection between reflective memories.**

Since the ORB provides a communication infra-structure, one can replace every communication path (including data reflection between reflec- tive memories) in the implementation of RT- CRM by the ORB GIOP/IIOP path. While the ORB GIOP/IIOP provides reliable communication, it incurs overhead at both the GIOP/ IIOP layer and TCP layer which GIOP/IIOP is based on. From our experimental results, in order to achieve performance similar to that of MidART but under CReMeS, we found that we need to use a separate data channel for data reflection between two reflective memories which does not make use of the ORB GIOP/ IIOP path. To this end, we make use of a separate UDP channel for this path. It should be pointed out that we employ the ORB GIOP/IIOP for setting up a RT-CRM service since a reliable communication path is desirable in this case. This approach of adopting separate communica- tion paths in CORBA environments is also seen in CORBA audio/video service specific- ation [14]. There, for the configuration and con trol of streams, it uses the GIOP/IIOP of the ORB. Once the streams are configured for an audio/video service, a different communication channel, such as UDP, is used for data stream transmission. However, there exists a trade-off between gaining performance by adoption of a separate UDP channel and losing reliability due to not using the ORB IIOP.

In order to achieve the third design goal, the design consideration that we faced can be described as follows. In CORBA environments, the location of the target CORBA object is supposed to be transparent to a client. A client usually contacts a naming server and obtains the object references to the target CORBA object. How can we achieve location transparency of the RIO if the (target) CORBA object is located in the same address space as the client? Our solution to this conundrum is to place the (target) RIO in the (dynamic) library to be linked with the application. When the library is initialized, the servant of the RIO is instantiated inside the library, registered into the ORB, and the object reference to the CORBA object is created and registered into a naming server. The library to be linked with the application at compile-time has information about the servant of RIO, but the location information is hidden from the applica-tion. An application is given the IDL of the interface at compile-time (and also the library to be linked), and then it contacts the naming server at run-time and obtains the object reference of the CORBA object. One copy of RIO is created for each application process and registered with a unique name. The uniqueness of the name is achieved by using process identifier as a part of the name. Thus, applications using CReMeS will have the same usage semantics that typical CORBA applications do.

## 4.2 Components of CReMeS

CReMeS is composed of three basic modules: the RT-CRM Server, a Library and the RT-CRM Service Coordinator. The *RT-CRM Server* is responsible for managing reflective memories, scheduling messages and dispatching them to the other RT-CRM Servers. The structure of the RT-CRM Server contains a scheduler, a dispatcher, and an admission control module. The scheduler employs dual priority[2] user-level scheduling scheme in order to integrate non real-time data as well as real-time data by providing fair bandwidth for non real-time data in such a way that real-time communication is not affected. The detailed str- uctures of scheduler, dispatcher, and admission control are the same as those in MidART (for details, the readers shou-ld refer to [6]). The RT-CRM Server exposes two CORBA objects, "Lib_Comm" and "CRM_Comm". Through the interface of "Lib_Comm" CORBA object, the library can deliver setup messages to the RT-CRM Server. "CRM_Comm" CORBA object provides an interface to the RT-CRM Server through which the other RT-CRM Servers can deliver setup messages when necessary. The RT-CRM server runs as a process in a node, one copy per node.

The *library* supports mapping to RT-CRM for applications (producers or consumers). An important component of the library is the servant of RIO discussed in the previous subsection. The library has the necessary support to allow RIO to setup the shared memory area between an application and the RT-CRM Server. Also, the library is responsi- ble for locating and calling the interface of the CORBA object "Lib_Comm" that allows the transfer of data between an application and the RT-CRM Server.

*RT-CRM Service Coordinator* provides a set of administrative services such as registration, and location of RT-CRM services for the entire system. Its functionality and structure are basically the same as those of the MidART's Global Server. It is implemented as a single process throughout the entire system, and exposes one CORBA object ("Glob_Comm") to RT-CRM Servers. This object is used by the RT-CRM Server to request the RT-CRM Service Coordinator for administration (such as registration and location) of RT-CRM services.

### 4.3 CReMeS Communication Structure

In CReMeS, three different kinds of communication paths are supported.

1. A path to setup a RT-CRM service: All communications for setup pass through the ORB GIOP/IIOP path. The three modules (the library, the RT-CRM Server and the RT-CRM Coordinator) use this communicati- on path to set up a RT-CRM service.
2. A path for data stream communication between an application (producer or consumer) and the associated reflective memory (in the same node): Producers write to their reflective memory through the shared memory mechan- ism. Similarly, making use of the shared memory mechanism, consumers read from the reflective memories in the RT-CRM Server in the consumer's node.
3. A path for data stream communication between producer's reflective memory and consumer's reflective memory: Data streams written into the reflective memory are scheduled and dispatched by the scheduler and dispatcher within the RT-CRM Server, and are copied or "reflected" into the reflective memory in the RT-CRM Server in the consumer nodes through a UDP channel (or the ORB GIOP/IIOP path).

## V. Experiments and Performance Evaluation

We have done extensive experimentation to evaluate the real-time properties of CReMeS and its ability to integrate real-time and non-real-time tasks. This section presents some of these results and compares the performance of CReMeS to TAO's Real-Time Event Service [7]. We begin with an overview of the experimental setup and a description of the metrics used in the evaluation, and then turn to the performance results.

### 5.1 Experimental Hardware and Software

All of the experiments were carried out on Pentium II PCs with Windows NT 4.0 operating systems over 100BaseT Fast Ethernet. We have isolated the network segment from the rest of the LAN while running all the experiments. The two PCs we used in the experiments have processor speeds of 333MHz and 266MHz respectively. The faster machine is equipped with 128MB of RAM and the slower machine has 64MB of RAM.

We ran the experiments described in this section on two real-time communication services, CReMeS and the Real-Time Event Service v1.9 which is included in the distribution of TAO v1.1.

The current implementation of CReMeS uses a robust, high-performance ORB, omniORB2 v3.0.0, developed by AT&T [16]. This ORB implements the specification 2.3 of the Common Object request Broker Architecture (CORBA). It supports the C++ language binding, is fully multi-threaded and comes with a COS Naming Service. We evaluated the CReMeS architecture under its two data communication mechanisms: one using an out-of-band channel (UDP) and the other using ORB GIOP/IIOP.

TAO's Real-Time Event Service is an object-oriented, real-time implementation of the CORBA Event Service intended to decouple suppliers and consumers and allow asynchrono- us event delivery in a predictable manner. RT Event Service runs on top of TAO, a real-time ORB end system that provides end-to-end quality of service guarantees [7].

## 5.2 Metrics and Experimental Setup

We evaluate the benefits of our real-time communication service as well as TAO's RT Event Service by measuring their *performance* and *predictability* as a function of load. For this purpose, we measure the *latency* and *jitter* introduced by the communication service under different load conditions. In addition, we also determine how well the communication service *scales* with respect to the sizes of messages using the service.

We used two simple applications that we believe are indicative of the type or real-time load that is likely to be present in a distributed industrial control system [9]. The following is a description of these applications:

- *Producer of sensor data*: This application represents a data acquisition device, such as a Remote Terminal Unit or Programmable Logic Controller, which gathers data from sensors and at regular intervals sends information to an operator workstation.
- *Consumer of sensor data*: This application represents an operator's workstation that processes the data arriving from the sensors and displays the results on the screen. In all of our experiments, this does not perform any processing of the data.

In our experiments we ran the above applications on two nodes connected via an isolated network. In one node, the *producer* application can be configured to run 1 to N threads, each thread representing a different device. In the other node, the *consumer* applic- ation creates 1 to N entities to receive the messages. Each *producer* thread, $P_0$ to $P_N$, sends messages through the communication service to its corresponding *consumer* entity, $C_0$ to $C_N$, in the other node. In our experime nts, we vary the size of message, the produc- tion rate of messages and the number of producer/consumer pairs to evaluate the real- time properties of the communication channel in question.

## 5.3 Latency and Jitter Measurements

The end-to-end latency is an important performance measure for any real-time commu- nication service. High latency overheads intro- duced by the communication service have a negative impact in meeting deadlines. However, high variance in the latency has a more detrimental effect for real-time applications that require end-to-end predictability. With this in mind, this experiment is designed to measure the latency and variance in the latency of our communication service. For this purpose, we configured the experimental setup described in the previous section in the following manner:

The producer application creates $n$ threads for each run. *Producer* thread $P_0$ sends a message to its corresponding consumer $C_0$ every 50 msec and it is assigned the highest priority. The other producers, $P_0$ to $P_n$, send a message to their corresponding consumers, $C_0$ to $C_n$, every 100 msec and are assigned a lower priority than $P_0$.

Both application programs first create their application threads and set up the necessary associations between producer and consumer by contacting the communication service. All threads block on a signal until all of the associations between producers and consumer have been accepted and established by the communication service. The highest priority thread, $P_0$, is responsible for releasing all other threads. Once the signal to run is given, threads are dispatched according to their priority and the communication service is responsible for the scheduling of messages between the two nodes.

In this experiment, each *producer* of data includes in its message a timestamp. Upon receiving the message, a *consumer* returns this timestamp to producer via a separate UDP channel in order to determine the round trip latency of each message. At the end of the experiment the average, maximum and minimum latency is calculated. The *jitter* is determined by calculating the standard deviation of the latency for a particular run. In each run the highest priority producer sends 4000 messages and the lower priority producers send 2000 messages. We vary the load in the experimental set up by increasing the number of low priority producer/consumer pairs from 0 to 50 and the size of the messages from 1KB to 8KB. Each messages is a sequence of values of type CORBA::Octet.

Figures 4 and 6 show the average latency produced by the high priority producer/ consumer pair for each of the communication services evaluated (CReMeS using the out band channel, CReMeS-ORB using omniORB2 and TAO RT-Event Service) for message sizes of 1KB and 8KB respectively. The jitter results forthe high priority producer/consumer pair with 8KB messages for all the communications services are shown in Figure 8. Due to space limitations, we

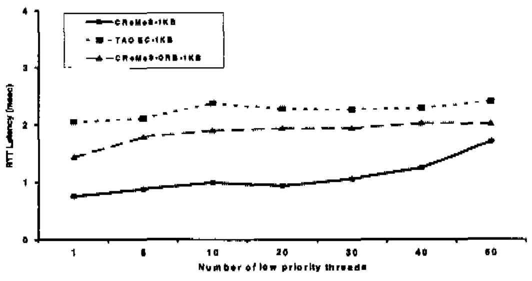


Fig. 4 Latency for Low Priority thread, 1KB

do not present the jitter results for messages of size 1KB. However, it is important to note that the jitter observed for all three communications services under all load conditions is less than 0.2 msec for the high priority thread and less than 1 msec for the low priority threads.

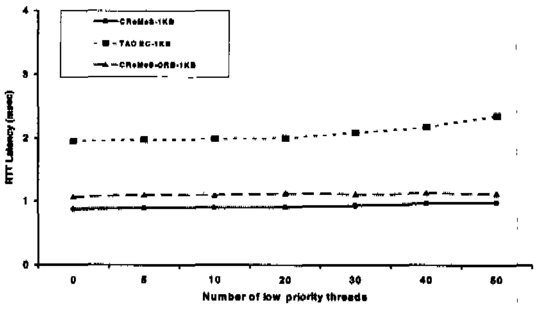


Fig. 5 Latency for High Priority thread, 1KB

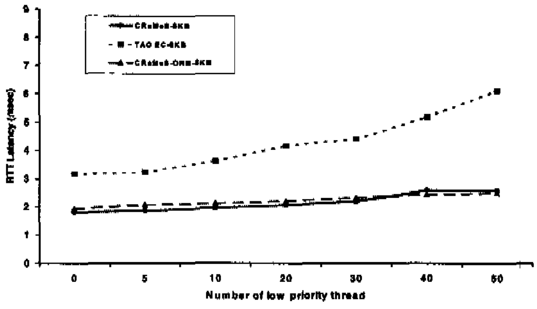


Fig. 6 Latency for High Priority thread, 8KB

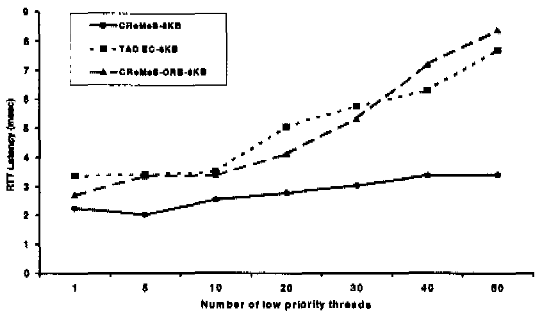


Fig. 7 Latency for Low Priority thread, 8KB

Figures 5 and 7 show the average latency produced by one of the low priority producer/ consumer pairs for each of the communication services for message sizes of 1KB and 8KB respectively. The jitter results for low priority thread with 8KB size messages are shown in Figure 9. Overall, CReMeS performed better than the rest of the services tested:

- CReMeS has a lower latency, under most load conditions, for both the high priority and the low priority producer/ consumer pairs than the other services.
- Perhaps, more importantly the variance in latency for CReMeS is quite low and remains stable as the load increases.

As Figure 5 shows, for CReMeS, latency values for the high priority producer range from 0.88 msec with no low priority producers to 0.99 msec (a 12% increase) with 50 low priority producers for 1KB. It ranges from 1.8 to 2.6 msecs (a 42% increase)
for messages of size 8KB.

- For all load conditions the jitter observed for the high priority thread when using CReMeS is less than the one observed using the RT-Event Service and similar to CReMeS-ORB. Figure 8 shows that jitter for the the high priority producer/consumer pair ranges from 0.025 to 0.3 msec for 8KB messages.

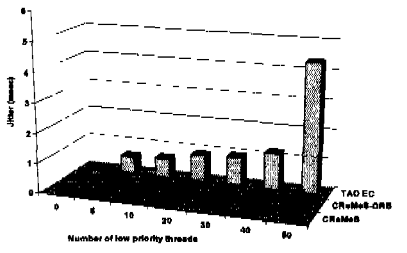


Fig. 8 Jitter for High Priority thread, 8KB

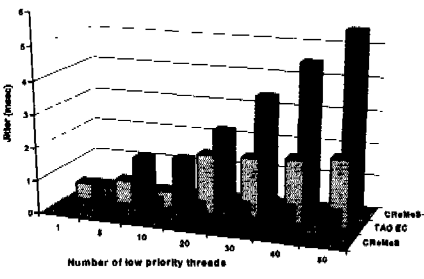


Fig. 9 Jitter for Low Priority thread, 8KB

- The jitter for the low priority producer/ consumer pair is higher than the one recorded for the high priority pair. However, as Figure 9 shows, the jitter variations for 8KB messages for the low priority pair are smaller for CReMeS than the ones produced by other services.

The low variation in jitter for both high priority and low priority producer/consumers pairs observed with CReMeS for different loads is an indication of the high level of *predictabil- ity* and *scalability* that CReMeS is capable of providing. A predictable and scalable service is useful to an application designer, since it allows him/her to determine a realistic bound for the worst case execution time incurred by the communication service. Figure 10 and Figure 11 clearly illustrate the above point.

Figure 10 shows a trace of the latency measurements taken for the high priority producer/ consumer pair when 40 low priority producers are active and sending messages of 8KB. Figure 11 shows a similar trace for the same load, but instead of using CReMeS as the communication service TAO's RT-Event Channel is used.

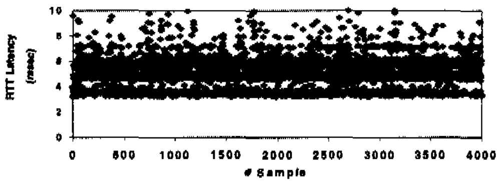


Fig. 10 Latency for RT-Event Service

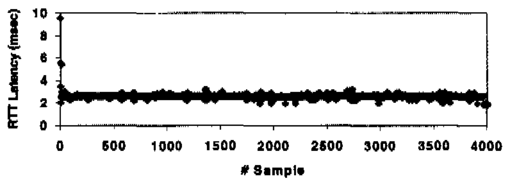


Fig. 11 Latency trace CREMeS

Figure 12 presents the average latency for the high priority thread with 40 low priority threads for each communication as a function of the size of the message. Similary, in Figure 13 the average latency for one of the 40 low priority threads for the same load, 40 threads, is shown for the same messages sizes. Clearly, CReMeS is the only communication service that scales, in terms of message size, for both high and low priority producers.

Turning to the specifics of the performance of the other threads, TAO's RT Event Service results indicate that this service is capable of providing predictable performance only for small messages or a small number of producer/consumer pairs. Figure 4 shows that the latency for the high priority producer increases 20%, from 1.9 to 2.3 msec, for 1KB size messages.

An increase in latency of 93%, from 3.1 msec to 6.1 msec, is observed in Figure 6 for the high priority producer/ consumer pair with 8KB messages as the number of low priority producers increases from 0 to 50. The percentage of increase in latency for both messages sizes is higher than the increase recorded for both CReMeS and CReMeS-ORB. Similarly, the latency for the low priority producer shows a similar increase of 17% and 128% for 1KB and 8KB message sizes respectively.

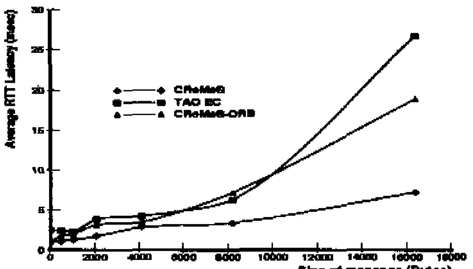


Fig. 12 Latency for Low Priority thread

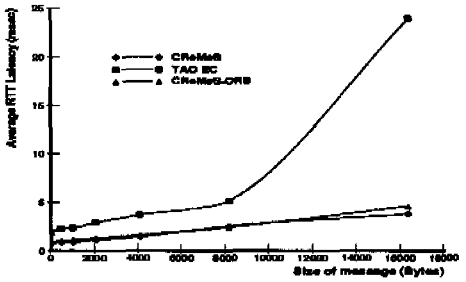


Fig. 13 Latency for High Priority thread

The jitter results for the RT Event Service also show a similar trend, however the increase is more pronounced for the high priority producer/ consumer pair, as shown in Figure 8. The jitter for the low priority producer/ consumer pair, shown in Figure 9, increases considerably once the number of low priority producer/consumer pairs is greater than 10, but remains constant, ranging from 1.8 msec to 2 msec from then on. These results indicate that TAO RT Event Service scales poorly on our test platform. However, our experimental results show that the RT Event Service performance is acceptable when the size of the message is low and the amount of traffic passing through the event channel is moderate.

CReMeS-ORB provides similar latency and jitter as CReMeS for the high priority producer/ consumer pair as CReMeS and similar to TAO's RT Event Channel for the low priority pair. The results indicate that CReMeS -ORB consistently provides lower latency than the RT Event Channel for most of the loads evaluated, as shown in Figures 4,5,6 and 7. However, the results presented in Figure 9, indicate that this communication service introduces more jitter for the low priority producer/ consumer than the other two service for most of the load conditions. One possible reason for this is that CReMeS-ORB uses the two-way synchronous request, instead of the one-way operations used by RT Event Service. As with the RT Event Service, this variation in jitter once the load increases does not bode well for systems that require predictable QoS.

## Ⅵ. Conclusions and Related Work

In this paper, we have described our experience with the design, implementation, and evaluation of CReMeS, a CORBA compliant communication service based on the concept of a Real-Time Channel-based Reflective Memory abstraction. CReMeS was designed as an alternative communication mechanism to the Event Service in order to provide adequate end-to-end QoS.

We have designed and implemented this service and have tested our implementation under various load conditions. Our experimental results demonstrate that our architecture can achieve better performance (measured by latency), predictability (measured by jitter) and scalability (measured in terms of number of clients or size of messages handled) that a real-time implementation of the CORBA Event Service when an out-band-channel is employed. This results are important for applications that required asynchronous communication of large size data. It is to be noted that these results were obtained even though CReMeS was implemented on top of an ORB which has not been optimized for real-time applications.

We designed a variation of CReMeS to use the ORB as the communication channel for the reflective memories, instead of UDP. Our results indicate that CReMeS-ORB consistently provides lower latency than the RT Event Service. We plan to conduct further experiments and modifications in this architecture to improve its performance. For some applications this variation may be desirable since it enhances the reliability of the communication because of ORBs use of TCP. As part of our future work, we will focus on the dispatching mechanism for the RT-CRM Server module. Currently, CReMeS employs a single dispatching thread. In [22], we have shown that a single dispatcher thread can minimize priority inversion, enable priority tracking and helps in dealing with limited operating system priority levels. However, this mechanism works well when the sending thread is not blocked after passing a message to the network layer. This is not the case in our current implementation of CReMeS -ORB, which may account for the difference in performance between CReMeS-ORB and the RT Event Service under heavy load conditions. One of future direction is to include a multi-threaded dispatching mechanism in CReMeS-ORB that works in unison with our current implementation of a dual priority user level scheduler.

We would like to conclude the paper with a discussion of some related work. In distributed real-time application environments, several research efforts to design and implement communication subsystem (middleware) have been reported. ARMADA [1] is a middleware supporting fault-tolerance and end-to-end guarantee for embedded real-time distributed applications, but unlike CReMeS, is not constructed on a standard interface. Rajkumar et.al [18] propose Real-Time publisher/subscriber model. This model is similar to event service and the RT-CRM communication paradigm used in CReMeS in that they all provide anonymous and group communication. But one key difference between the RT-CRM communication paradigm and many of the current publisher-subscriber communications models is that RT-CRM effectively achieves the decoupling of the writer/publisher's quality of service (QoS) characteristics from the reader/subscriber's QoS requirement. This decoupling allows much more flexibility in constructing distributed and/or concurrent real-time applications. What this means is that RT-CRM is not simply a multicast or unicast data transport protocol. It is not restricted to transmit the data to the receivers/readers immediately after the data is made available by a writer. In contrast, it provides application designers the facility to specify how and when the data should be sent according to the application's specific needs.

Recently, OMG adopted a Notification Service [15] which extends the COS Event Service by adding event filtering, and event delivery QoS to event service functionalities. However, this Notification Service will have the same limitation as COS Event Service since it employs the same event channel architecture as COS Event Service does.

In [8], TMOSM (TMO Support Middleware) is developed as a TMO(Time-triggered Message-triggered Object) execution engine in CORBA environments. TMOSM is adapted from original TMO execution engine to be used in CORBA environments just as our CReMeS is adapted from original MidART. Enabling or enforcing QoS capabilities for CORBA to make CORBA adaptable for QoS required applications has been vigorously pursued by many researchers. DARPA Quorum program [3] and associated projects [5,24] are representatives of these research activities. Quorum program tries to build flexible and adaptive real-time distributed object middleware for mission-critical systems. QoS requirements considered in this program include mission-critical attributes such as constraints, dependability, and some security attributes as well as performance measure such as bandwidth, latency, and jitter. We believe that our work will contribute to the goals of this program.

## References

[1] T. Abdelzaher, and et. al, "ARMADA Middleware and Communication Services," *The International Journal on Time-Critical Computing Systems*, vol. 16, no. 2/3, pp. 127-153, 1999, Kluwer Academic Publishers.

[2] R. Davis and A. Wellings, "Dual Priority Scheduling," *Proc. of 16th IEEE Real-Time Systems Symposium*, Dec., 1995.

[3] DARPA:1999, "The Quorum Program", *http://www.darpa.mil/ito/research/quorum/index.html*

[4] P. Felber, B. Garbinato, and R. Guerraoui, "The design of a CORBA group communication service," *Proc. of the 15th IEEE Symposium on Reliable and Distributed Systems (SRDS'96)*, Canada, Oct., 1996.

[5] W. Feng, U. Syyid, and J.S. Liu, "Providing for an Open, Real-time CORBA," *Proc. of the Workshop on Middleware for Real-time systems and Services* , San Francisco, CA., 1997.

[6] O. Gonzalez, C. Shen, "Implementation and Performance of MidART". *Proc. of IEEE Work-shop on Middleware for Distributed Real-Time Systems and Services*, San Francisco, CA., Dec., 1997.

[7] T. Harrison, L. Levine and D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," *Proc. of OOPSLA '97*, ACM, Atlanta, GA, Oct., 1997.

[8] K.H. Kim, M. Ishida, and J. Liu, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation," *Proc. of IEEE CS Int'l Symp. On Object-Oriented Real-time distributed Computing*, 1999.

[9] K. Ramamritham, C. Shen, O. Gonzalez, S. Sen and S. Shirgurkar, "Using Windows NT for Real-Time Applications: Experimental Observations and Recommendations," *Proc. of 4th IEEE Real-Time Technology and Applications*, Denver, Colorado, June, 1998.

[10] Object Management Group, The Common Object Request Broker: Architecture and Specifica tion, *OMG Document 99-10* .

[11] Object Management Group, CORBAservices: Common Object Services Specification, *OMG Document 98-12*.

[12] Object Management Group, CORB Messaging Specification, *OMG TC Document orbos/98-05-05*

[13] Object Management Group, Real-Time CORBA Joint Revised Submission, *OMG Document ptc/99-05-03*.

[14] Object Management Group, Control and management of Audio/Video Streams, *OMG RFP Submission, OMG TC Document telcom/97-05-07*.

[15] Object Management Group, CORBA Notification Service, *OMG TC Document telcom/99-07-01*.

[16] omniORB2, *http://www.uk.research.att.com/omniORB*.

[17] C. O'Ryan, Douglas C. Schmidt, D. Levine, and R. Noseworthy, "Applying a Real-time CORBA Event Service to Large-scale Distributed Interactive Simulation," *Proc. of 5th International Workshop on Object-oriented Real-Time Dependable Systems (WORDS '99), IEEE*, Monterey, CA., Nov., 1999.

[18] R. Rajkumar, M. Gagliardi, and L. Sha, "The Real-Time Publisher/Subscriber Interprocess Communication Model for Distributed Real-Time Systems: Design and Implementation," *Proc. of 1st IEEE Real-Time Technology and*

*Applicat- ion Symposium*, May, 1995.

[19] D. C. Schmidt, D. Levine, and S. Mungee, "The Design of the TAO Real-Time Object Request Broker," *Computer Communications Special Issue on Building Quality of Service into Distributed Systems*, Elsevier Science, Volume 21, No. 4, April, 1998.

[20] D. C. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers," to appear, *The International Journal on Time-Critical Computing Systems*, 2000, Kluwer Academic Publishers.

[21] C. Shen and I. Mizunuma, "RT-CRM: Real -Time Channel-based Reflective Memory," *Proc. of 3rd IEEE Real-Time Technology and Applica- tions Symposium*, Montreal, Canada, June, 1997.

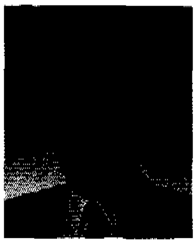
[22] C. Shen, O. Gonzalez, K. Ramamritham and I. Mizunuma, "User Level Scheduling of Communicating Real-Time Tasks," *Pro. of 5th IEEE Real-Time Technology and Applications Symposium,* Vancouver, Canada, June, 1999.

[23] V. Wolfe, L. Dipippo, R. Ginis, M. Squadrito, S. Wohlever, I. Zykh and R.Johnston, "Expressing and Enforcing Timing Constraints on a Dynamic Real-Time CORBA System," *The International Journal on Time-Critical Computing Systems,* 1999, Kluwer Academic Publishers.

[24] J.A. Zinky, D.E. bakken, and R. Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice for Object Systems,* Vol. 3, no.1, 1997.

[25] MidART User Guide, *http://www.merl.com/projects/midart/index.html*

정 선 태(Sun-Tae Chung) 정회원



1983년 2월 : 서울대학교 전자공학과 졸업(학사)
1985년 12월 : 미국 미시간 대학교(앤아버) 전기 및 컴퓨터공학과 석사
1990년 12월 : 미국 미시간 대학교(앤아버) 전기 및 컴퓨터공학과 박사
1991년 3월~현재 : 숭실대학교 정보통신전자공학부 부교수
<주관심 분야> 실시간 시스템, 분산 시스템, 임베디드 시스템, 디지털 제어