MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

RT-CRM: Real-Time Channel-based Reflective Memory

Chia Shen, Ichiro Mizunuma

TR97-01 April 1997

Abstract

In this paper, we propose and present Real-Time Channel-based Reflective Memory (RT-CRM) – a useful programming model and middleware communication service for constructing distributed real-time industrial monitoring and control applications on commercially available open systems. RT-CRM provides remote real-time data reflection abstraction using a simple writer-push model. This writer-push approach enables us to easily decouple the QoS characteristics of the writers from that of the readers. This decoupling is crucial in supporting different kinds of remote data transfer and access needs that one often finds in distributed industrial systems. We will describe the design of RT-CRM, along with a set of easy-to-use API to access the RT-CRM service. We have implemented RT-CRM as part of a larger real-time middleware project, MidART. We address many of the important implementation issues including buffer management and QoS support. We demonstrate the feasiblity of RT-CRM through a discussion of our application programming support and preliminary performance data.

Proc. IEEE Third Real-Time Technology and Applications Symposium (RTAS97), Montreal, Canada, June 9-11, 1997

RT-CRM: Real-Time Channel-based Reflective Memory

Chia Shen

MERL - A Mitsubishi Electric Reseach Lab.
201 Broadway
Cambridge, Massachusetts 02139
USA

Ichiro Mizunuma

Industrial Electronics and Systems Lab.
Mitsubishi Electric Corp.
8-1-1, Tsukaguchi-honmachi
Amagasaki, Hyogo, 661, Japan

Abstract

In this paper, we propose and present Real-Time Channel-based Reflective Memory (RT-CRM) – a useful programming model and middleware communication service for constructing distributed real-time industrial monitoring and control applications on commercially available open systems. RT-CRM provides remote real-time data reflection abstraction using a simple writer-push model. This writer-push approach enables us to easily decouple the QoS characteristics of the writers from that of the readers. This decoupling is crucial in supporting different kinds of remote data transfer and access needs that one often finds in distributed industrial systems. We will describe the design of RT-CRM, along with a set of easy-to-use API to access the RT-CRM service. We have implemented RT-CRM as part of a larger real-time middleware project, MidART. We address many of the important implementation issues including buffer management and QoS support. We demonstrate the feasiblity of RT-CRM through a discussion of our application programming support and preliminary performance data.

1 Introduction

The availability of high speed networks, such as ATM which can support QoS sensitive real-time communication [12], and the availability of real-time operating systems on PCs and workstations, such as Lynx or QNX, as well as real-time scheduling support in general purpose operating systems, such as those found in IRIX and Solaris, have lead to increased interest in both industry and academia to design distributed real-time systems using open, standard, commercially available computers and networks. Meanwhile, to facilitate the construction of distributed real-time applications on such open systems, we must first provide easy-to-use real-time programming models and services to the real-time application designers. In this paper, we propose and present Real-Time Channel-based Reflective Memory (RT-CRM)[1] — a useful programming model and middleware communication service [11] for constructing distributed real-time industrial monitoring and control applications on such commercially available open systems. We have designed and implemented RT-CRM as part of an ongoing project *MidART* – *Mid*dleware and network *A*rchitecture for distributed *R*eal-*T*ime industrial systems [7]. MidART provides a set of real-time client-server computing facilities for building industrial applications.

Figure 1 is an example of the application environment we are considering. The characteristics of the class of distributed industrial applications for which RT-CRM is designed for, and their implications for the requirements of the underlying real-time system and communication support are:

(1) Data sharing and communication patterns are often unidirectional. E.g., plant data is sent from the plant controller to the operator stations, and control data is sent from the operator stations to the plant controllers. Therefore, interaction between the operators and the controlled system can be decoupled.

(2) Not all the data need to be periodically broadcasted to all the nodes in the network all the time. Typically, there are many producers/writers, each producing their respective separate plant data or video, while a consumer/reader will need data from a subset of these producers. The members of this subset are not fixed and will change from time to time whenever a reader requests to do so. In general, a LAN-based industrial plant has hundreds of plant control sensors (i.e., writers), but

[1] RT-CRM actually should be read as Real-Time *and* Channel-based Reflective Memory, and is not based on real-time channels [4], although real-time channels can be one of the underlying communication support to implement RT-CRM.

only a handful, usually 5 to 10, operator stations (i.e., readers). Therefore, the dominant type of data distribution can be viewed as many-to-one in nature,instead of the common one-to-many multicast model.

(3) Historical data are often requested by the operators. This data enable the operators to review past plant activities, e.g., in the form of a trend graph of temperature sensoring data. This plant history data should be retrieved in real-time and with response time small enough to support interactive display of past and present sensory information. This characteristic has two important implications. The first is that this kind of history data is needed only for the recent past, but may be requested very frequently, thus the main memory should be used for storing the history data. The second implication is that since the sensory data are not immediately "consumed", we need to support both the constant data generation from the controllers and the frequent reviewing of historical data simultaneously.

(4) QoS (Quality of Service) in terms of bounded message delay on plant data updates and control message delivery are required. The delay bound requirements usually range from a few milliseconds to a couple hundreds of milliseconds.

(5) Plant controllers are simple computers with limited computation and storage capacity, while operator stations (OPS), data loggers and multimedia servers have the capability to perform sophisticated functions, and have more memory and disk capacity.

While most of the current research focuses on real-time message communication, as examplified in [4, 6, 10], after analyzing the characteristics of distributed industrial applications, we have approached the problem from a memory-to-memory data transfer perspective. This enabled us to devise a useful real-time distributed programming model, and to provide a set of intuitive middleware services. The concept of RT-CRM is based on four principles:

- To provide data reflection with guaranteed timeliness to distributed real-time applications. We define *data reflection* as the memory-to-memory data transfer between remote hosts in a networked environment.
- To provide flexibility in how and when data are reflected.
- To keep the servers (let it be an industrial plant controller, or a multimedia server) as simple as possible — only to perform the necessary data reflection.

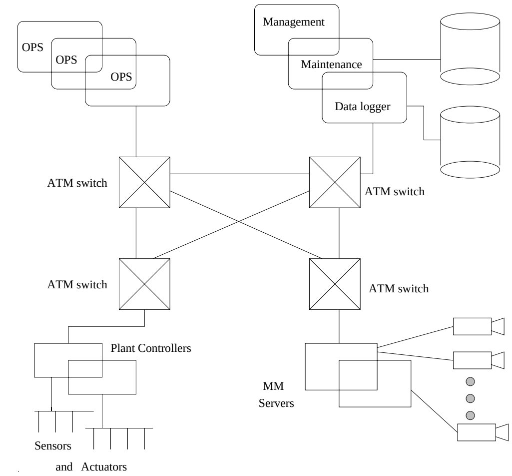

Figure 1: An Example Industrial Plant System with ATM Networks. OPS = Operator Stations.

- To enable the construction of a distributed industrial system in a plug-and-play fashion, and to give the application designer an easy-to-use interface.

RT-CRM supports these principles based on two key properties: (a) a 'writer-push' data reflection model, and (b) the decoupling of writers' and readers' QoS. The simplicity of a writer-push data reflection model makes it easier to provide predictability with flexibility in the fashion (e.g., synchronous vs. asynchronous) in which the data are reflected onto remote nodes. This should be attractive to industrial applications. Moreover, it also supports video transmission naturally. One should be able to use RT-CRM for both traditional data/control communication, as well as multimedia (video, audio, image) communication. The writer-push model also enables many higher level functions, such as displaying the history of plant monitoring data or setting control values, to have simple designs where most of the computation only occurs on the reader's node.

There has been a lot of real-time research addressing the issue of providing end-to-end delay guarantee. End-to-end in a networked environment can mean many things to an application. We classify end-to-end into three different levels – Application-to-application (AtA), memory-to-memory (MtM), and network interface-to-network interface (NtN). AtA is where the guarantee is provided from the moment the sending application generates the data to the moment the receiving application retrieves the data. MtM is where the guarantee is provided from the moment when the data is taken from the sending host memory to the moment when the data is deposited into the receiving host memory, regardless of when the data is generated by the sending application and when the receiving application actually retrieves the data. NtN is simply the network guarantee from when the data is transmitted from the sending network interface to when the data is entirely received by the receiving network interface. We have discovered that different application scenarios require different levels of end-to-end guarantee. We have designed RT-CRM to allow the application to choose between AtA and MtM according to its own requirement[2].

We have implemented the first prototype of RT-CRM over an ATM LAN with one FORE Systems ATM switch connecting PCs running QNX as the operator workstations and plant controllers. Our preliminary performance tests show that RT-CRM incurs very little overhead and is a feasible solution for real-time plant monitoring and control applications.

The rest of the paper is organized as follows. We discuss related remote memory systems and their limitations in Section 2. Section 3 gives the detailed architecture design of RT-CRM. We have implemented a set of API which provide programming access to the RT-CRM middleware service. These APIs and application programming support are also described in Section 3. In Section 4 we address important implementation issues of RT-CRM. These include (1) concurrency control and synchronization, and (2) buffer management schemes and a proof for the minimum number of buffers needed to avoid locking for readers of the reflective memory. Section 5 discusses QoS and network interface support issues that are both closely related to the design and implementation of RT-CRM. Performance comparisons with IP via socket interface are shown in Section 6. The paper concludes in Section 7.

2 Related Work on Remote Memory Systems

In this section, we discuss existing remote memory systems[3]. In particular, we point out their limitations for supporting the type of industrial applications under consideration.

[2]Note that with a good network interface hardware technology such as those found in [8], one can narrow the gap between MtM and NtN. However, this is beyond the scope of our work reported here.

[3]DSM, Reflective Memory and Memory Channels, at some level, are all systems and protocols which allow reads and writes on physically distance memories in a networked environment. Thus we call them *Remote Memory Systems*. DSM is a higher level protocol than both Reflective Memory and Memory Channels, but still provides remote memory services.

2.1 Why Not Distributed Shared Memory?

Distributed shared memory provides transparent reads and writes of shared data in a networked environment. However, we do not need the full semantics of a DSM system such as those in TreadMarks DSM system described in [2]. Most of the functionalities of a DSM system are built to provide an illusion of a global virtual memory and to support concurrent writes on different nodes, e.g., a read must return the value that is last written. Thus, a DSM system must implement functions to deal with (1) managing local process page faults while the physical page last written is on a remote site, (2) coherency protocols, such as invalidation for replicated copies, (3) consistency model, e.g., sequential consistency, eager or lazy release consistency.

The distributed industrial plant control application domain does not require this full set of DSM semantic support. For example, we do not need the invalidation process at all. Our data in general is updated either periodically, or upon a change of value. In either case, the reader usually can read the latest copy on its local processor. Synchronization only needs to occur when the local copy is being actually updated. More importantly, full DSM support will magnify many worst case delay bounds for data updates where multiple writers/multiple readers issue writes and reads. In a real-time system, we must consider this worst case delay.

2.2 Why Not Reflective Memory?

Hardware supported reflective memory, such as what is provided by SCRAMNet or VME Microsystems, Inc. [13], replicates (or reflects) data in all nodes of the network in a bounded amount of time (e.g., 1usec/node latency). These reflective memory systems are based on a ring topology, can only support a limited physical memory size, typically 1 to 16 MByte, and a limited number of nodes (up to 256). These hardware reflective memory systems are very expensive. Since we do not need to distribute data to all the nodes all the time, reflective memory will greatly limit the amount of actual memory we can support. For example, for a N-node system, we need $K * N$ system memory assuming each node needs to reflect data of size K.

2.3 Why Not Memory Channels?

Memory Channel is a hardware-software combined technology from Digital Equipment Corp., originally licensed from Encore Computer Corp [5]. It is designed for low latency high performance clustered parallel computing and is in the middle ground as far as performance and scalability are concerned between symmetric multiprocessors and ATM. A Memory Channel is shared. Reads and writes on a Memory Channel are supported directly by DEC's PCI-MC adaptors. For writes, the adaptor can send writes to a single node, or multiple nodes, on per page basis. Reads are supported via non-swappable physical memory — the adaptor can DMA incoming data with a known shared address space map into the corresponding physical memory.

Memory Channel can only support a limited number of nodes (up to 8 AlphaServers), and a limited distance (3-meter link from the Memory Channel Hub to a server). Although Memory Channel is a useful concept, we need to support potentially up to hundreds of nodes in a network, and we need data updates (i.e., data reflection) with specifiable time bounds and frequency.

In summary, our Real-Time Channel-based Reflective Memory is much more flexible compared with either hardware supported reflective memory and memory channels, or software supported distributed shared memory. In hardware supported reflective memory, the data is reflected *immediately* in a bounded amount of time to other nodes as soon as the writer application deposits its new data. In distributed shared memory, the new data is made available to other readers or writers with one of two methods: either upon data release, or upon data acquisition. There is no time constraint guarantee associated with either of these methods. In RT-CRM, we allow the reader applications to specify *when* it wants the data to be reflected and we guarantee the timeliness of this reflection using a real-time writer-push model.

3 RT-CRM: Design, API and Application Programming Support

3.1 Overview of RT-CRM

In Real-Time Channel-based Reflective Memory (RT-CRM), we combine the benefits from (1) Reflective Memory (i.e., updates propagated in bounded amount of time), (2) Memory Channel (i.e., hardware assisted, virtual connection based memory to memory transfer of data), and (3) open standard ATM networks. The *unidirectional* access pattern and bounded update reflection time of the applications require *reflective*, rather than, *shared* memory semantics. To eliminate the lack of scalability problem in traditional Reflective Memory, we use the concept of channels. Specifically, ATM enables us to provide flexibility in channel establishment and cost reduction.

In a distributed real-time monitoring and control system, we require applications to specify, at memory

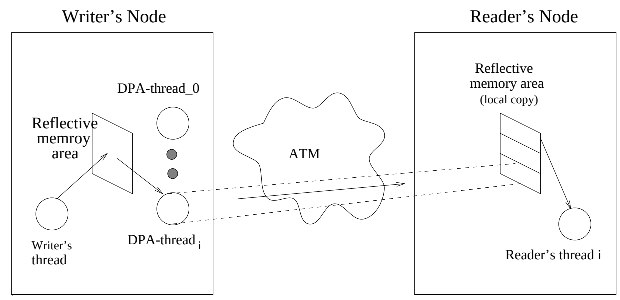

Figure 2: RT-CRM High Level Architecture

channel establishment time, (1) *who* needs the data, and (2) *when or how often* a reader needs the data. The schedulability or admissibility of read and write operations can be determined. This allows RT-CRM to use a writer-push (vs. a reader-pull) underlying model in which data produced remotely will be actively pushed through the network and written into a reader's memory without the reader explicitly requesting the data at run time.

Figure 2 depicts the high level architecture of RT-CRM. RT-CRM is an association between a writer's memory and a reader's memory on two different nodes in a network with a set of protocols for memory channel establishment and data update transfer. A writer has a memory area where it stores its current data, while a reader establishes a similar memory area on its own local node to receive the data reflected from the writer. Data reflection is accomplished by a data push agent thread, a DPA-thread, residing on the writer's node and sharing the writer's memory area. This agent thread represents the reader's QoS and data reflection requirements. A virtual channel is established between the agent thread and the reader's memory area, through which the writer's data is actively transmitted and written into the reader's local memory area. In this architecture, we support the following features:

- A reader memory area may be connected to multiple remote writer memory areas simultaneously. However, at any moment only one writer is permitted to write into the reader's memory area via the associated agent thread.
- A writer memory area may be connected to many remote reader memory areas simultaneously. There can be many data push agent threads representing many readers associated with the same writer memory area.

These features enable us to satisfy the application requirements described in Section 1, yet minimize the complexity of the design on the writer's node. The writer only needs to deposit its data into the designated memory area, while all the other more compli-

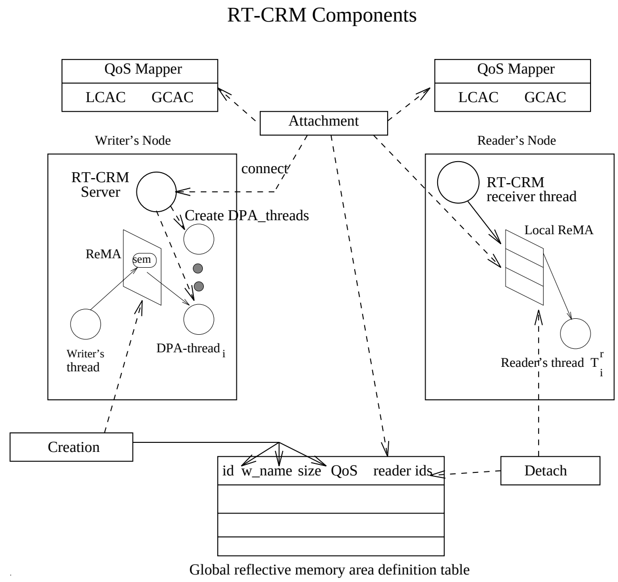

Figure 3: Reflected memory area, threads, and system tables in RT-CRM.

cated operations and QoS support are handled by the data push agent threads and the readers. In essence, RT-CRM is a distributed programming service provided in MidART. Many other more sophisticated or useful application functions, such as histories of data, continuous video and video alarm, can be built on top of this service.

3.2 Detailed Design of RT-CRM

Figure 3 illustrates the key components and operations of RT-CRM. Discussions throughout this section will refer to the figure. A RT-CRM consists of:

(1) a reflective memory area (ReMA) owned by the writer node,
(2) a set of QoS parameters,
(3) a semaphore (sem) with priority inheritance,
(4) a writer thread that updates the reflective memory area periodically according to the writer's QoS,
(5) one or more data push agent threads, DPA_threads, one for each reader connection, defined by readers QoS parameters, and
(6) a set of one or more readers, each has a local copy of the ReMA.

Creation

We allow a ReMA to be created by either a writer or a reader. This flexibility is necessary to support a LAN based industrial environment where nodes may join and leave dynamically, and new plant data may be requested to be added into the system by any node. At creation time, each reflective memory area is associated with a global id, a size, QoS in terms of update period/frequency and a semaphore for read-write conflict resolution on the writer's node. This information is initialized in the reflective memory area definition table. The table is a network-wide global table, allowing all potential readers and writers to know what is the QoS/period of the writer for this reflective memory area[4]. DPA-threads are created when readers request attachment to this reflective memory area. In the case when a ReMA is created by a reader, the QoS will be replaced by a writer's QoS later. The global definition table will be updated when other information regarding a particular ReMA becomes available.

Mapping and Attachment

Once created, a reader can "attach" itself to a ReMA by allocating a corresponding reflective memory area on the reader's local node and associating these two remote reflective memory areas. The reader's real-time data reflection requirement is specified as (a) periodic (with or without a deadline), (b) upon every data update, or (c) conditional (i.e., when some condition X becomes true). The reader's period or minimum interarrival time must be greater than or equal to the update period of the writer's reflective memory area. The attachment to an ReMA includes the following actions:

(1) The reader specifies its data reflection QoS requirement (i.e., type (a), (b), or (c) as described above).
(2) The reader also specifies the number of past data copies (i.e., history) H it requires.
(3) Upon receiving a reader's request, the following must be done:
    (3.1) The schedulability on both the reader's and writer's nodes must be examined by LCAC and GCAC (Local and Global Connection Admission Control). LCAC on the reader's node examines whether the data reflection QoS requested by the reader can be scheduled on the reader's local CPU. Similarly, the LCAC on the writer's node must check the schedulability of the DPA-thread with the QoS requested by the reader on the writer's CPU. The reader's QoS must be equal to or less strict than that of the writer's. Meanwhile GCAC examines the schedulability of the data reflection QoS requirements on the network. The attachment

[4]This network-wide global table is only a logical design. To address bottleneck and reliability issues, the actual implementation can be distributed — One option is for each node to keep a local copy of the table and a separate control channel is set up such that table updates can be broadcast to all nodes. Since table updates do not occur frequently, this will not result in wasted usage of network bandwidth.

of a reader to a ReMA can be admitted into the system only when both LCAC and GCAC are successful. We discuss the algorithms used for LCAC and GCAC in Section 5.

(3.2) Sets up a connection between the reader and the writer according to the reader's QoS.

(3.3) Allocates a circular buffer area of size = N * the size of one reflective memory area. This circular buffer is shared (or mapped to) between the application and the network interface (where the network interface supports direct memory access [8] by the interface card), or between the application and a RT-CRM receiving thread (where the network interface does not support direct memory access by the interface card). Note that this set of N buffers is allocated on the reader's machine. (How to determine the minimum value of N will be described in Section 4.2.)

(3.4) Creates a data push agent thread, DPA_thread, on the writer's node on behalf of the reader. This thread will either be a periodic thread or a thread waiting on a signal. Once activated (either periodically, or upon an update signal), this DPA_thread will lock and read the reflective memory area, unlock, and transmit the data over the established VC.

Similarly, a writer can "map" itself to a ReMA. If the ReMA has been created by a reader, this mapping includes allocating a corresponding reflective memory area on its own node.

Since a reader's real-time data reflection requirement can be specified as (a) periodic, (b) upon every data update, or (c) conditional (i.e., when some condition X becomes true), DPA_threads can also be of three types respectively. DPA_threads of type (a) are asynchronous with respect to the writer's thread, while DPA_threads of type (b) and (c) are synchronous. If the reader requires data reflection of types (b) or (c), the corresponding DPA_thread may be signalled whenever the writer thread completes a write operation in the reflective memory area. Since the periodic writer thread is an application thread, it should only do the write operation in a critical section and then releases the lock. Whether the writer thread should evaluate conditions to activate any DPA_threads (for type (c) data reflection) depends on the specific application. For example, if the writer is associated with an operator's command task, then the writer thread should wake up the DPA_thread to transmit the operator's commands. On the other hand, if the writer is a periodic sensor, there may be many DPA_threads reading/waiting on the associated reflective memory area. Then we do not want to force the writer's thread to take the responsibility of evaluating conditions and signalling all the waiting DPA_threads.

3.3 Support for Application Programming

Our design of the RT-CRM with an underlying writer-push model and the DPA-threads allows us to decouple how data is updated on the writer's node from how the data is reflected to the reader. Given a reflective memory area, since a DPA-thread is a separate thread of control from the writer's application thread, the DPA-thread can either push the data to the reader's node synchronously or asynchronously with respect to the write operations conducted by the writer's thread. In particular, RT-CRM supports the following types of data push and read operation modes:

- Data Push Operations:
  - *Synchronous Data Push*: Pushes are triggered by application writes.

    When the writer's application thread does a write in the reflective memory area, the DPA-thread sends/pushes the contents of the reflective memory area to the reader/receiver immediately or conditionally. This can be implemented with a signal to the DPA-thread from the writer's thread.
  - *Asynchronous Data Push*: Pushes are performed periodically.

    The DPA-thread sends the contents of the reflective memory area to the reader periodically, i.e. with independent timing from that of the writer's application.
- Read operations:
  - *Blocking Read*: Application reads block while awaiting the arrival of a data update from the writer's node. When the new data is received, the reader application thread is signalled.
  - *Non-Blocking Read*: Application reads return the current contents of the reflective memory area. That is, the reader's application thread will not be notified upon the arrival of data update messages.

With this set of data push and read operation modes, we can support at least four combinations for application programming as listed in Table 1. In the table, the Combination column lists the possible data-push and read operation mode. With respect to each type of Combination, Data Transmission shows the corresponding traffic that will be generated into the network, Delay Bounds defines what level end-to-end QoS guarantee RT-CRM must provide, and Application Example gives the potential usage of the Combination. For example, to implement remote operator command issuing, one can use the combination of SB, i.e., as soon as the operator enters a control command, the corresponding DPA-thread will be signaled to push the command data to the appropriate plant controller, while blocking read is used on the plant controller computer to receive the remote command. This combination provide Application-to-Application delay guarantee. On the other hand, one can imagine situations where only Memory-to-Memory delay guarantee is required. In these cases, the application designer can choose the non-blocking read mode.

3.4 Application Programming Interface

We provide sixteen basic interface functions for applications to access RT-CRM services. Table 2 lists the API of RT-CRM. Most of the API are intuitive. Below we discuss a few that contain special features.

`CRM_Create` creates a reflective memory area entry in the global reflective memory area definition table. A globally unique id for this reflective memory area is returned in `m_id`. The value of `m_mode` can be either **shared** or **exclusive**. If `m_mode` is set to **shared**, more than one local thread can map this reflective memory area into its address space and thus become the writer of the reflective memory area. To allow different threads to map to the same memory area will allow the application threads to be upgraded/modified/replaced at any time without having to re-establish network connections, or to re-create DPA-threads. In this way, RT-CRM can become the plug-and-play interface points. Also, allowing more than one local asynchronous threads to access/write into the same memory area provide flexibility to writer applications. If two application threads want to reflect their values to the same reader, they can do so. On the other hand, there might be applications that would like to restrict the number of writers of a reflective memory area to be only one for security or safety reason. Then the value of `m_mode` should be set to **exclusive**.

If the reflective memory area has not been created yet (this would be the case if the ReMA has been created by a reader), `CRM_Map` creates a reflective mem-

//* Creates a reflective memory area by making an entry in the global reflective memory area definition table. *//
`CRM_Create(int m_size; int m_w_period; void *m_addr; int m_mode; int m_id)`
//* Removes the reflective memory area identified by `m_id`. It also terminates all of the DPA-threads and the network channel connections to the readers associated with this memory area. *//
`CRM_Destroy(int m_id; void *m_addr)`
//* Allocates a reflective memory area of `m_H` buffers. Maps the reflective memory area pointed to by `*m_addr` to the calling thread. This memory area must have been created with the the value of `m_mode` equal to **shared**. *//
`CRM_Map(int m_id; void *m_addr; int m_H)`
//* Tears down the mapping between the calling thread and the reflective memory area. *//
`CRM_Unmap(int m_id)`
//* Allocates a reflective memory area of`m_H` number of buffers if `*m_addr` is null. Attaches a reader's thread to the reflective memory area and establish network connections with the writer's ReMA. *//
`CRM_Attach(int m_id; int m_H; int m_r_period; int m_deadline; void *m_addr; int DR-FLAG)`
//* Detaches a reader thread from the reflective memory area by removing the associated DPA-thread and its connection. *//
`CRM_Detach(int m_id)`
//* Activates the associated DPA-thread on the writer's node for the calling reader thread. *//
`CRM_Start(int m_id)`
//* Fills the reader's buffers with existing data from the writer's buffers. *//
`CRM_StartInitH(int m_id; void *m_addr)`
//* Halts the reflection of the memory area by suspending the associated DPA-thread. *//
`CRM_Stop(int m_id)`
//* Read a single memory buffer. By definition, it will be the most recent available data. *//
`CRM_Read(int m_id; void *m_addr)`
//* Read `h` buffers counting back from the most recently updated buffer. *//
`CRM_ReadH(int m_id; void *m_addr; int h)`
//* Read all available data in the buffers. `howmany` returns the number of data buffers read. *//
`CRM_ReadAll(int m_id; void *m_addr; int howmany)`
//* Writes the data pointed to by `*m_data` into the memory area pointed to by `*m_addr`. *//
`CRM_Write(int m_id; void *m_addr; void *m_data)`
//* Locks the memory area for exclusive use. Priority Inheritance must be enforced here. *//
`CRM_Lock(int m_id)`
//* Releases the lock of the memory area after exclusive use. *//
`CRM_UnLock(int m_id)`
//* Resets all the contents of the reflective memory area. *//
`CRM_Reset(int m_id)`

Table 2: RT-CRM_ Application Programming Interface

Combination	Data Transmission	Delay Bound Required	Application Example
SB	Sporadic	AtA	Command issuing
AB	Periodic	AtA	Trend graph
SN	Sporadic	MtM	Plant data
AN	Periodic or Sporadic	MtM	Video or file transfer

Table 1: S = Synch., A = Async., B = Blocking, N = Non-blocking, AtA = Application-to-Application, MtM = Memory-to-Memory.

ory area of `m_H` buffers with each buffer equal to the size specified in the global reflective memory area definition table, then maps the reflective memory area pointed to by `*m_addr` to the calling thread. If the reflective memory area already exists, then the calling thread must reside on the local node where the reflective memory area is allocated, and this memory area must have been created with the the value of `m_mode` equal to **shared**. With this library function, we allow a reflective memory area to have more than one local writer threads.

As we described at the beginning of the paper, usually an operator station would like to be able to monitor a subset of the plant controllers, and at times operators switch the membership of this subset. In particular, to optimize the usage of memory, we would like to use the same memory buffer on the reader's node to potentially receive data reflected from different writers. The API functions `CRM_Attach`, `CRM_Start`, `CRM_Stop`, and `CRM_StartInitH` support this flexibility. With `CRM_Attach`, a reader can use the same local memory area to attach to different remote reflective memory areas. When the reader needs to switch from the data reflected from one writer to that of another, `CRM_Stop` will halt the reflection of a memory area by suspending the associated DPA-thread on the current writer's node, and `CRM_Start` will activate the associated DPA-thread on the other writer's node for the calling reader thread. Then `CRM_StartInitH` will fill the reader's buffers with existing data from the new writer's reflective memory area.

4 Implementation Issues

In this section, we address a few important implementation issues. These include concurrency control for read and write operations to the same reflective memory area, buffer management schemes and QoS guarantees.

4.1 Concurrency Control and Synchronization

For predictability, we strictly impose writer-push for updates. Reflective memory has always been writer-pushed. This also is useful for video transmission. All locks/semaphores are local. All *read* operations are local in nature even though the writer/owner of the reflective memory area is remote. That is, we do not need to deal with remote reads and page faults. Each reader has an agent thread on the writer's machine representing the reader, called the data push agent thread (DPA-thread). This thread performs the read locally on the writer's machine, and then sends the read value via the network into the reader's address space on the reader's machine.

On a writer's node, we use lock-based concurrency control between the writer's thread and all the readers' DPA-threads for potential read-write conflicts. One single semaphore is used for one-writer-multiple-reader access of a particular reflective memory area on the writer's machine. The semaphore state should be set to priority inheritance to avoid unbounded wait of the writer when one or more readers are waiting simultaneously for the semaphore. This way, we can ensure that the writer will not be blocked more than one reader's critical section since the writer has a higher priority than all the other reader threads. For scalability, reads with the same QoS should be grouped together as one read operation. On the reader's node, no locking is needed. Concurrency control is done via sufficient buffer replication as described below.

4.2 Buffer Management on Reader Nodes

In this section, we describe the details of how the buffer space on the reader's node is managed in RT-CRM for correct and efficient data reflection.

4.2.1 Overview

Upon a reader's request to attach to a specific reflective memory area D, we allocate a circular buffer area of size $= N * DS$ that is shared (or mapped) between the reader application and the network interface (or a message receiving thread), where DS is the size of the reflective memory area D. (See Figure 4.) This set of N buffers is allocated on the reader's machine. Among the set of QoS parameters provided by the reader for its reflective memory area attachment, the amount of past data history required by the reader is specified by H. The value N is calculated from H. If the reader needs a maximum H of past data, N will be equal to

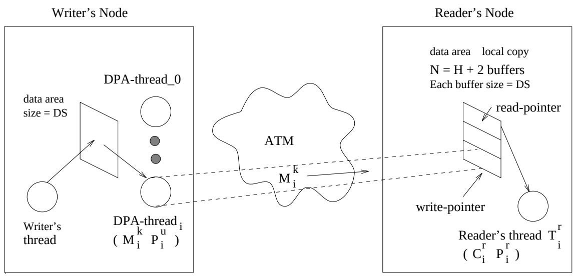

Figure 4: Terminology Illustration for Buffer Management.

$H + 2$. For example, if the reader only needs a single copy of the reflective memory area (i.e., the most recent available data), N will be equal to 3. This buffer allocation scheme simplifies the design – *we do not need locking on the reader's node.* A *write-pointer* is used that points to the next buffer area in the circular buffer that the next new incoming data will be written into. Then the reader application can always read the buffer area until it reaches the buffer just before the *write-pointer*. This design also supports history or other types of higher level applications (e.g., video transmission) that need to read more than one buffer at a time.

The design of the buffer management for RT-CRM includes a proof of the minimum value for N, and a concrete design that uses only the minimum number N of buffers, including the specification of a set of primitive operations for reading and writing the buffers, and the implementation details of a set of basic API. These are described in the following sections.

4.2.2 Minimum Value of N

Locking restricts concurrency and also is an expensive operation, especially if it is required for every read and update operation. Thus we would like to avoid using locks as much as possible in our design. In this section, we prove that $N = H + 2$ is the minimum number of buffers that is necessary and sufficient to avoid locking each buffer for concurrent reads and/or updates on the reader's machine under the assumptions discussed below. Since *MidART* is a distributed real-time system, to prove the minimum value for N, we must reason about the worst case scenario between the reads and updates, taken into consideration the minimum interarrival time of the reads and the updates, as well as the worst case network jitter that the update messages will incur.

Terminology, Assumptions and the Worst Case Scenario

DPA_thread_i is a thread on the writer's machine to reflect data to the reader according to the reader's QoS. Thus DPA_thread_i will transmit a data update message M_i^k to the reader's node either periodically, or with a minimum interarrival time. Let P_i^u be the period or the minimum interarrival time of the messages from DPA_thread_i for an application reader thread T_i^r. Remember that P_i^u has already been guaranteed for DPA_thread_i when DPA_thread_i was created at the time the reader attached itself to the reflective memory area. Let C_i^r be the worst case computation time of the operations in thread T_i^r that (1) calculate the index to a buffer to be read next, and (2) read the buffer. Thus, with respect to the particular circular buffer used by T_i^r on the reader's node, P_i^u is the period of the writes.

We assume that $C_i^r < P_i^u$. This is a very reasonable assumption since reading the contents of a local data buffer should require less time than the time required to (1) transmit the same amount of data over the network, and (2) writing the data into the data buffer.

Due to network queueing and cell scheduling jitter, any data update message M_i^k from DPA_thread_i can experience a maximum network delay of $D_{M_i^k}^{max}$ and a minimum network delay of $D_{M_i^k}^{min}$. If M_i^k incurs $D_{M_i^k}^{max}$, and M_i^{k+1} only incurs $D_{M_i^k}^{min}$, then the two updates from M_i^k and M_i^{k+1} will be *back-to-back*, i.e., M_i^k will update the data buffer at the end of the current P_i^u while M_i^{k+1} will update the next data buffer at the beginning of the next P_i^u. This is the worst case scenario we must guard against when a reader is reading a buffer to avoid race conditions.

We prove the following theorem for the minimum number of buffers needed to allow concurrent reads and writes into the circular buffer without locking.

Theorem 1:
$N = H + 2$ buffers are necessary and sufficient to ensure that concurrent reads and writes are not issued to the same buffer.

Proof
We will prove for the case of $H = 1$ since it implies the general case of for all $H > 1$. That is, we will prove for $N = 3$.

Necessary: Since we need at least two buffers to accommodate the worst case when updates from two messages M_i^k and M_i^{k+1} arrive *back-to-back* from the network in any two consecutive P_i^u periods, we must have a third buffer for reading concurrently. Thus the *necessary* part is obvious.

Sufficient: Let the buffers be indexed by $I = \{1, 2, 3\}$, and a *write-pointer* always points to the buffer that is either currently being updated, or is the next buffer to be updated if no update operation is in progress. Assume that reads and writes always proceed from buffer I to buffer $(I+1) \bmod 3$. In this protocol, the read starts at (*write-pointer* + 2) mod3.

Suppose that the current *write-pointer* is pointing to buffer 1, and a read starts in buffer 3. Since we know that $C_i^r < P_i^u$, then even in the worst case when a *back-to-back* update occurs — as soon as the read starts, the write into buffer 1 completes and the *write-pointer* is incremented to buffer 2 — we are still guaranteed that the read in buffer 3 will finish before the write to buffer 2 can complete. Thus $N = 3$ is sufficient. □

4.2.3 A Design with $N = H + 2$ Buffers

In this section, we first give a concrete design of a circular buffer with a set of associated primitive operations. Then we will show how to use the design to implement the associated API functions in RT-CRM.

We define a circular buffer area as a memory area allocated on the reader's node and consisting of (see Figure 5):

- N reflective memory area buffers, each reflective memory area buffer is of size DS, where $N \geq 3$,
- an index I for each buffer, where $I = \{1, 2, ..., N\}$,
- a *write-pointer* that always points to the buffer which is either currently being written into (i.e., being refreshed), or is the buffer to be written into next if there is no write operations in progress, and
- a *read-pointer* that points to the buffer that is currently being read.

Below is the set of protocols for primitive read and write operations that must be followed.

- All read and write operations are always performed in the direction of increasing values of $I \bmod N$.
- **Start read**
  - If I is the index of the buffer that the *write-pointer* is pointing to, then the start read operation will return the buffer indexed by I' where

$$I' \geq (I+2) \bmod N \quad \text{if } N > 3$$
$$I' = (I+2) \bmod N \quad \text{if } N = 3$$

- **Stop read**

  The read operation always stops at the buffer indexed by $I'' = (I-1) \bmod N$ where I is the buffer pointed to by the *write-pointer*.

Algorithm M_Read
begin
Read write-pointer I;
Read buffer ((I-1) mod N);
end

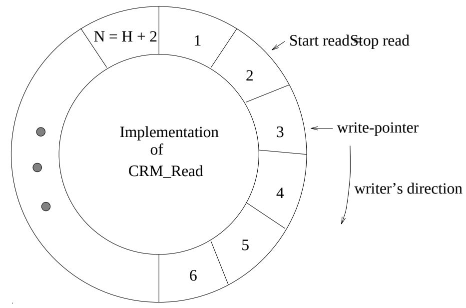

Figure 5: Reading the most recent reflective memory area.

Algorithm M_Read_History
begin
Read write-pointer I;
I' = ((I + 2) mod N) + (H-h);
for i = 1 to h do
Read buffer I';
I' = I' + 1;
end

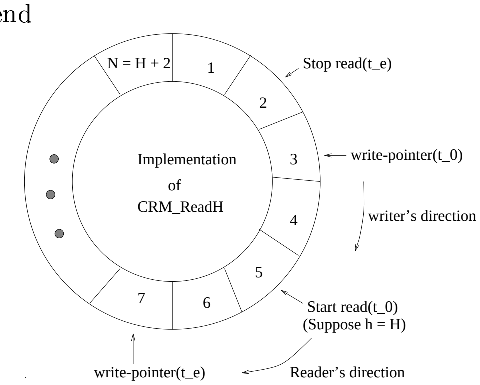

Figure 6: Reading a history of size h.

Note that if only the most recent data is to be read, then **Start read** will be the same as **Stop read**. In the case of $N = 3$, one should notice that $(I+2) \bmod N$ is the same as $(I-1) \bmod N$. Thus for $N = 3$, we always read the buffer that is two away from the *write-pointer*. However, when $N > 3$, we must use the calculation in **Stop read** for reading only the most recent data buffer.

- **Write**
  - Write the new data into the buffer I pointed to by the *write-pointer*, and move the *write-pointer* forward such that $I = (I+1) \bmod N$.

Algorithm M_Read_All
begin
 Read write-pointer I;
 I' = (I + 2) mod N;
 while I' not equal to ((I - 1) mod N) do
 Read buffer I';
 I' = I' + 1;
end

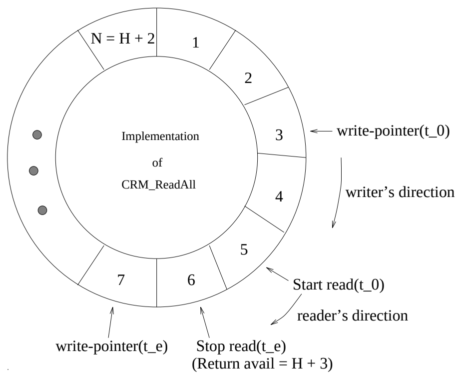

Figure 7: Reading all available history up to now.

Algorithms *M_Read*, *M_Read_History*, and *M_Read_All* implement the API functions `CRM_Read()`,`CRM_ReadH()`, and `CRM_ReadAll()` respectively. The algorithms are presented in pseudo code shown in Figures 5, 6, and 7. In Figures 6 and 7, t_0 is the time when the read operations start, and t_e is when the read operations complete.

5 Discussion

Although we do not directly address the problem of network interface design, and the problem of QoS guarantee algorithms for the host system and the network in this paper, these are issues that closely influence how RT-CRM achieves its goals.

5.1 Network Interface Support

The RT-CRM architecture on the reader side can be implemented in two ways depending on the type of network interface hardware and software available.

(1) With direct memory deposit capability from the network interface such as those discussed in [8], we do not need a receiving thread in the middleware on the reader's node. Upon receiving a message with the newly updated data from a DPA-thread, we can identify the memory area/buffer address where the data update message should be written into, and do the correct calculation for the circular buffer indexes as described in Section 4.2. This will no doubt provide a much more efficient and low latency data reflection path.

(2) Without any direct memory deposit facility from the network interface, we will need to create a receiving daemon or driver thread on the reader's node. This daemon thread will be mapped into the same data area circular buffer memory as our application reader's thread. This is our current implementation since we do not yet have any network interface with direct memory deposit capability available.

5.2 End-to-End QoS Support

Although real-time task scheduling in the host system as well as network message transmission scheduling are orthogonal to the issues that RT-CRM addresses, RT-CRM relies on these underlying end-to-end scheduling mechanisms to guarantee the timeliness of the data push operations. There is a large body of research results on the subject of real-time task scheduling and real-time message communication. In particular, since the first target network for RT-CRM is ATM, we use ATM traffic class CBR which provides a constant cell rate service and bounded cell delay variation. Real-time communication can be supported by this traffic class with appropriate network switch scheduling [1, 14]. For scheduling the DPA-threads, writers as well as readers in the host systems, we use rate-monotonic scheduling algorithms [3, 9] with operating system support on QNX.

Moreover, in scheduling a writer thread and the DPA-threads associated with the same writer's reflective memory area, we use a Writer-QoS based correctness model — the writer has higher priority over readers (i.e., the DPA-threads). This is because the writer is usually constrained by either the physical plant control components (e.g., sensor sensing rate), or the operator's command issuing timing constraints. In either case, it does not make sense to give the writer a lower priority than the readers.

6 Performance

We have implemented the first version of RT-CRM on an ATM-based LAN environment. The host systems are Digital's VENTURIS FX (Pentium 133Mhz, PCI bus) PCs running QNX real-time operating system version 4.23. Since our version of QNX does not support POSIX threads, we implemented all the DPA-threads as processes. The network interface cards on the PCs are FORE Systems PCA-200ePC for PCI bus. We used one ATM switch, FORE Systems ASX-200BX, to connect the host systems. Since current ATM software does not support CBR and rt-VBR traffice classes in switched virtual channels, our implementation used PVCs. We would like to eventually implement RT-CRM using native ATM, but again currently for the first version, we must live with available commercial ATM software which only supports IP interface.

Tasks	Period (sec)	Priority	Mode
writer #1	1.00	19	
writer #2	0.50	21	
writer #3	0.20	23	
writer #4	0.10	25	
writer #5	0.05	27	
DPA #1	1.00	20	ADP
DPA #2	0.50	22	ADP
DPA #3	0.20	24	ADP
DPA #4	0.10	26	ADP
DPA #5		28	SDP
reader #1	1.00	19	NBR
reader #2	0.50	19	NBR
reader #3	0.20	19	NBR
reader #4	0.10	19	NBR
reader #5		19	BR
receiver		29	

Table 3: Task Parameters. (ADP = Async data push. SDP = Sync data push. NBR = Non-blk read. BR = Blocking read.)

We focused our performance tests on two aspects of RT-CRM. One is how much overhead RT-CRM really incurs compared with raw UDP/IP. The other is the delay in switching from one writer's ReMA to another writer's ReMA for a reader. Table 3 lists the tasks and their parameters we used in all of our experiments reported here. All the writer tasks and the DPA tasks reside on one host, while all the reader tasks reside on a remote host. The priorities of tasks are such that a higher number indicates a lower priority. Since it is very difficult to measure the overhead for one-way communication in a LAN environment without sychronized clocks, our measurements are all round-trip times. To do this, we must use synchronous data push operation mode and blocking read as described in Section 3.3. In particular, our overhead measurements were all done with respect to writer #5, DPA #5 and reader #5 in Table 3. In the performance results shown below, for each data message size, we did 100 runs on an unloaded system and network, and extracted the minimum, maximum and the average latencies.

Table 4 shows the performance of round-trip latency RTT. To compare the round-trip latency of RT-CRM with that of raw UDP/IP, each measurement includes the time executing the following steps:

- Writer #5 starts a timer and writes into the ReMA on its own local node.
- Writer #5 signals DPA #5.
- DPA #5 sends data to the reader host.
- A receiver task on the reader host receives the data and deposits into the ReMA.
- Reader #5 reads the data and sends an acknowledgement back to the writer's host to stop the timer.

The worst-case and average round trip time RTT in Table 4 is almost proportional to messsage size. And more importantly, most of the RTT is the overhead of IP/UDP itself. RT-CRM itself incurs very little extra overhead.

Table 5 shows the total latency in switching from one writer's ReMA to another writer's ReMA for a reader. This switching incurs two round-trip overhead cost. It requires a reader to send stop control signal (using the `CRM_Stop` call) to the current writer, and upon receiving an acknowledgement, the reader sends a start signal (i.e., `CRM_Start` call) to a different writer. Only after receiving the newly reflected data from the second writer, we stop the timer for measurement. This experiment tells us whether RT-CRM can support interactive Rt-CRM memory channel switch for plant operators. In this experiment, the reader makes a request to switch from writer #4 to writer #5 every 1.01 sec. We used such a period in order to avoid phasing problems between the switching requests and write operations in the writer node. Strictly speaking, the performance numbers show in Table 5 really includes the waiting time for the next period of writer #5, and therefore we should expect a difference between the min and max of about 50 msec (i.e, the period of writer #5). Thus the min values should be very close to the *pure* switching time of RT-CRM. Remember that the latency requirement for this switching operation in our application domain is the actual interactivity requirement of the human operator with the machines. The min values are definitely sufficient for human operator interactivity requirement.

msg size (byte)	max	min	avg
1	59.103009	9.136029	34.579086
512	58.420803	10.168557	34.222680
1024	60.126318	10.242309	35.609863
1536	62.495601	10.306842	36.405831
2048	60.089442	11.035143	35.974197
2560	61.675110	11.579064	36.941271
3072	62.255907	12.252051	37.623661
3584	61.739643	12.288927	37.094951
4096	67.298700	12.740658	38.838356

Table 5: ReMA Switching Latency. (Time is in msec.)

7 Conclusion

We have described in detail the design, API, implementation and preliminary performance results of

msg size (byte)	UDP(avg)	UDP(max)	RT-CRM(avg)	RT-CRM(max)	ratio(avg)	ratio(max)
1	2.710293	3.300402	2.977183	3.337278	1.10	1.01
512	3.172350	3.742914	3.545074	4.047141	1.12	1.08
1024	3.571846	4.572624	4.097753	5.319363	1.15	1.16
1536	3.895765	4.526529	4.516572	5.116545	1.16	1.13
2048	4.268332	5.678904	5.723155	7.153944	1.34	1.26
2560	4.660573	5.807970	5.806402	6.785184	1.25	1.17
3072	4.878141	6.075321	6.586975	8.121939	1.35	1.34
3584	5.190481	6.324234	6.737245	8.002092	1.30	1.27
4096	5674294	7.052535	7.611298	10.666383	1.34	1.51

Table 4: Round Trip Latency. (ratio = RT-CRM / UDP. Time is in msec.)

RT-CRM. We demonstrated how distributed industrial plant applications can utilize RT-CRM to facilitate its remote data reflection. Our preliminary performance shows that RT-CRM incurs very little overhead, and is a feasible solution for many application environment.

There are still a lot of work to be done to fully explore the potential advantage and lessons of using RT-CRM. First of all, since the high speed network technology is highly active and fastly evolving, we are examining whether RT-CRM is also applicable on other open standard high speed networks such as Switched Ethernet. Second, we believe that RT-CRM is a general real-time programming model, besides our implmentation on QNX, we are currently porting it onto Lynx, Linux and IRIX as well.

References

[1] A. Raha and S. Kamat and W. Zhao. Admission Control for Hard Real-Time Connections in ATM LANs. *Proceedings of the 15th IEEE INFOCOM*, March 1996.

[2] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared Memory Computing on Networks of Workstations. *IEEE Computer*, pages 18–28, February 1996.

[3] C.L. Liu and J.W. Layland. Scheduling Algorithm For Multiprogramming in a Hard-Real-Time Environment. *JACM*, 20(1), January 1973.

[4] D. Ferrari and D. Verma. A Scheme for Real-Time Channel Establishment in Wide-Area Networks. *IEEE Journal on Selected Areas in Communications*, 8(3), April 1990.

[5] Digital Equiqment Corporation. Memory Channel Overview. *www.unix.digital.com/bin/textit/cluster/memchanl/memchanl.html*, May 29 1996.

[6] A. Mehra, A. Indiresan, and K.G. Shin. Resource Management for Real-Time Communication: Making Theory Meet Practice. In *IEEE Real-Time Technology and Applications Symposium*, June 1996.

[7] I. Mizunuma, C. Shen, and M. Takegaki. Middleware for Distributed Industrial Real-Time Systems on ATM Networks. In *17th IEEE Real-Time Systems Symposium*, December 1996.

[8] R. Osborne, Q. Zheng, J. Howard, R. Casley, D. Hahn, and T. Nakabayashi. DART — A Low Overhead ATM Network Interface Chip. In *Hot Interconnects 96*, August 1996.

[9] R. Rajkumar. Synchronization in Real-Time Systems: A Priority Inheritance Approach. *Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061 USA.*, 1991.

[10] R. Rajkuma, M. Gagliardi, and L. Sha. The Real-Time Publisher/Subscriber Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation. In *IEEE Real-Time Technology and Applications Symposium*, May 1995.

[11] S. Bocking. Sockets++: A Uniform Application Programming Interface for Basic-Level Communication Services. *IEEE Communications Magazine*, 34(12), December 1996.

[12] C. Shen. On ATM Support for Distributed Real-Time Applications. In *IEEE Real-Time Technology and Applications Symposium*, June 1996.

[13] VME Microsystems International Corporation. Reflective Memory Network. *White Paper*, February 1996.

[14] Q. Zheng, T. Yokotani, T. Ichihashi, and Y. Nemoto. Connection Admission Control for Hard Real-Time Communication in ATM Networks. In *Submitted to 17th International Conference on Distributed Computing Systems*, 1997.