

# Two CORBA Services Enabling TMO Network Programming

Kane Kim

Electrical & Computer Engineering Dept.  
University of California  
Irvine, CA, 92697  
khkim@uci.edu

Eltefaat Shokri

SoHaR Incorporated  
Beverly Hills, CA  
shokri@sohar.com

**Abstract:** In facilitating efficient construction of real-time distributed computing applications composed of CORBA-compliant components, there are practical advantages in keeping the core component of the CORBA standards, i.e., ORB and interface definition language (IDL), unchanged or minimally changed. This means to bring new mechanisms and capabilities for real-time computing support in the form of CORBA services. This paper discusses how this "minimal-extension" strategy can be realized. We focus here on the design and implementation of a middleware providing execution support for the *time-triggered message-triggered object* (TMO) structured CORBA applications. The TMO structuring scheme is a syntactically simple and natural but semantically powerful extension of the conventional object structuring approaches. The strategy discussed here is to provide the additional capabilities for executing and connecting TMO-structured CORBA-compliant components via two CORBA services, TMO Execution Support (TMOES) and Cooperating Network Configuration Management (CNCM). Both services can be implemented in the form of distributed computing middleware consisting of CORBA-compliant objects.

**Keywords:** real time, CORBA, object, TMO, Time-Triggered Message-Triggered Object, distributed systems, network, programming, service, COTS.

## 1 Introduction

As distributed object technologies are maturing and thus becoming viable ways for design and implementation of large-scale complex distributed applications, their applicability for various application domains, such as real-time (RT) systems, are being investigated by both academic and industrial organizations. Specifically, the applicability of CORBA standards for RT distributed applications has recently been considered as an important issue among researchers and practitioners in the real-time distributed computing (RTdC) field [OMG95]. As a result, several RT extensions of the CORBA standards have been proposed

in which both the programming interfaces and the internal structure of the object request broker (ORB) were extended. However, we recognize that there are practical advantages in keeping the core component of the CORBA standards, i.e., ORB and interface definition language (IDL), unchanged or minimally changed and incorporating the necessary RTdC-related capabilities and domain-specific requirements as "additional CORBA services". This paper discusses how this "minimal-extension" strategy can be realized.

We focus here on the design and implementation of a middleware providing execution support for the *time-triggered message-triggered object* (TMO) structured CORBA applications. The TMO structuring is an extension of conventional object-oriented (OO) design and implementation techniques with the following features [Kim94, Kim97]:

- (i) Each TMO may have a new group of *time-triggered (TT-) methods* in addition to conventional *message-triggered service methods*,
- (ii) In the case of a potential data access conflict between a TT method and a message-triggered service method, the execution engine executes the TT- method in a manner free of interference by the service method execution, and
- (iii) A time-window is imposed for each output action and completion of a method.

To fully support the TMO structuring, the runtime system should provide, among other services, the following important capabilities:

- *Capability A:* timely activation and execution of each TT- method.
- *Capability B:* timely transmission of remote service-requests between client and server hosts.
- *Capability C:* timely recognition of the arrival of a service request (i.e., timely initiation of the execution of a message-triggered service method).
- *Capability D:* timely execution of each message-triggered service method

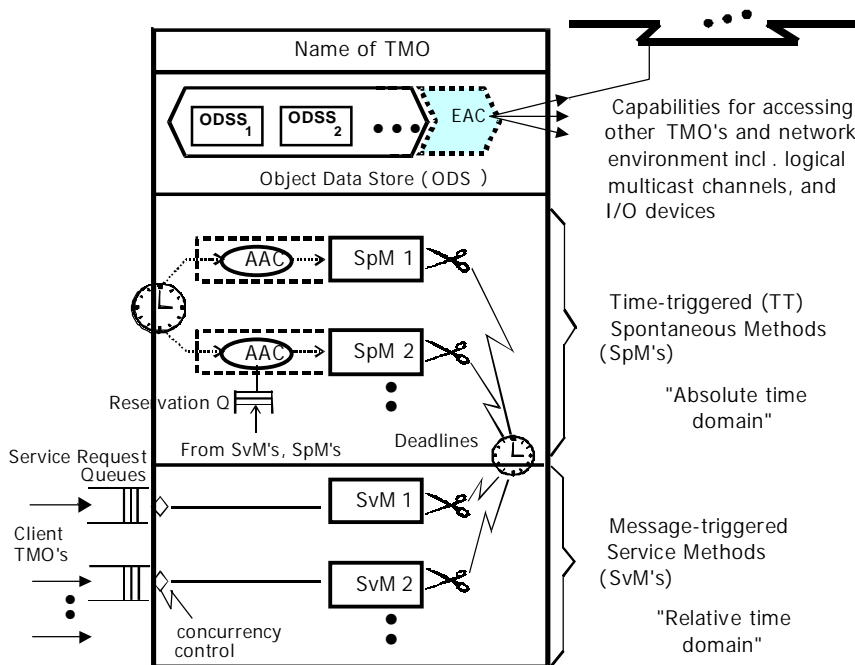


Figure 1: The basic TMO structure (adapted from [Kim97])

The support for capabilities A, C, and D can be provided as a middleware which "acts as a CORBA service" and does not require any change to the core ORB. This new service, named *TMO Execution Support (TMOES)*, can be implemented in the form of a CORBA-compliant distributed computing middleware. It is aimed for accepting the specifications of desirable timings of various components of a TMO and controlling local machine resources for accurate execution of time-sensitive components of the TMO with the cooperation of local operating systems on participating nodes.

To provide a fully predictable support for capability B, the operating systems together with the communication subsystems on relevant nodes should facilitate a predictable (RT) communication service. One cost-effective approach is to define another CORBA service through which a network of nodes with a certain desired level of guaranteed capacities for inter-node communication can be established and maintained. This new service may be called the *Cooperating Network Configuration Management (CNCM)*. CNCM service which runs in a distributed cooperating fashion in various nodes of a network, will maintain RT computing groups of cooperating nodes. Each group will be configured to maintain inter-node communication channels of the qualities desired by a chosen TMO network structured application.

The approach proposed in this paper is therefore to handle the special needs of RT applications (i.e., computation and communication predictability) through two CORBA services without any changes in the standard ORB.

The rest of the paper is organized in the following manner. Section 2 briefly introduces the TMO structuring scheme. In Section 3, our previous two ORB-vendor-dependent implementations of TMO support middleware are reviewed. Section 4 discusses in more detail the minimal-extension approach for supporting CORBA-compliant TMO-structured applications. Section 5 provides a conclusion of the paper.

## 2 Basic properties of TMO structuring

The *Time-triggered Message-triggered Object (TMO)* structuring scheme was established a few years ago [Kim94, Kim97] with a concrete syntactic structure and execution semantics for economical reliable design and implementation of real-time systems. TMO is depicted in Figure 1 and some of its major characteristics are as follows:

- (a) *Spontaneous method*: The TMO contains new types of methods, *time-triggered (TT-) methods*, also called the *spontaneous methods (SpM's)*, which are clearly separated from the conventional *service methods (SvM's)*. The SpM executions are triggered when the real-time clock reaches specific values determined at design time, whereas the SvM executions are triggered by service request messages from clients. Moreover, actions to be taken at real times that can be determined at the design time can appear only in SpM's.
- (b) *Basic concurrency constraint (BCC)*: Under this rule, SvM's cannot disturb the executions of SpM's and the designer's efforts in guaranteeing timely service capabilities of TMO's are greatly simplified. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place*. An SvM is allowed to

execute only if no SpM that accesses the same portion of the Object Data Store (ODS) to be accessed by this SvM has an execution time window that will overlap with the execution time window of this SvM. However, the BCC does not stand in the way of either concurrent SpM executions or concurrent SvM executions.

- (c) A *time window* is imposed for each output action and completion of a method of a TMO.

Extensions (a) and (b) are unique to the TMO structure in comparison with other proposed object extensions [Att91, Ish90, Tak92]. Client methods (SpM's or SvM's) may request the services of SvM's in other TMO's. To maximize concurrency in the execution of client and server methods, client methods are allowed to make non-blocking (sometimes called asynchronous) types of service requests to SvM's.

The designer of each TMO indicates the *time window for every output* produced by each SvM (and by each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) and advertises this to the designers of potential client objects. The designer of the server object thus guarantees the timely services of the object. Before determining the time window specification, the server object designer must make sure that with the available *object execution engine* (hardware plus operating system) the server object can be implemented such that the output actions are performed within the specified time-windows. Prototypes of efficient middleware running on widely used operating system platforms such as Windows NT and Sun Microsystem's Solaris to realize TMO execution engines have been built [Kim99,

Sho97].

### 3 Earlier approaches for supporting CORBA-compliant TMO network programming

Support for capability A (timely activation and execution of each TT- method) and D (timely execution of each message-triggered method) can usually be provided by a support middleware with no changes in the core ORB. However, capability C (timely initiation of the execution of an arrived service request) can be provided with or without changes in the core ORB and each approach has its own pros and cons. In this section, two approaches adopted in our earlier prototype implementations of a middleware architecture named TMOSM (TMO support middleware) [Kim99, Sho97, Sho98] are briefly mentioned:

- (a) *Vendor-specific interception approach*: In this approach adopted in our implementation of TMOSM on the Windows NT platform equipped with Iona's Orbix ORB, the filter capability of Orbix is used for intercepting service requests. The filter is represented by point A in Figure 2. The interceptor parses the arrived request, identifies the relevant server object and SvM, and informs TMOSM of the arrived SvM call. TMOSM ensures that the newly arrived SvM call will be executed in a timely manner. It should be noted that although the interception approach is used with Orbix, a similar approach can be used with several other ORB implementations.

- (b) *Integrated approach*: In this approach adopted in our implementation of TMOSM on the Windows NT platform equipped with OmniORB2, the function of associating the newly arriving service-request with the corresponding SvM and initiating the execution of the SvM is performed inside the modified core ORB immediately after the object adapter locates the server object. The point of recognizing the arrival of an SvM call is represented as B in Figure 2.

While being effective in specific environments, these approaches have some drawbacks such as "dependence on a specific ORB implementation" or "availability of the ORB's source code". Hence, they are not of the generic type and may not be readily used with different ORB implementations.

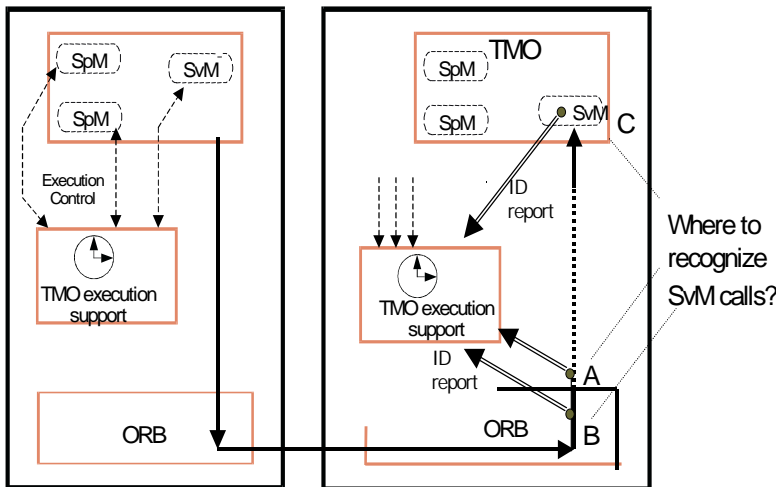


Figure 2. Recognition of SvM calls

#### 4. The CORBA service approach for supporting TMO's

Unlike the approaches discussed in Section 3, the proposed CORBA service approach possesses the following attractive characteristics:

- *No changes in the core ORB specifications and functionality:* One important philosophy of the OMG in creating the CORBA was that the ORB architecture should not depend on operating system platforms or any implementations techniques. This is the main reason why the OMG specifications do not identify or even recommend any implementation approaches. Moreover, the core architecture of the ORB is generic and does not pay special attention to any specific application domains. Unless the OMG produces standard specifications for domain-specific ORB's (e.g., real-time ORB, fault-tolerant ORB, secure ORB, etc.), any approach of tailoring the core ORB and its interfaces toward a specific application domain is incompatible with the goal of the OMG and should be avoided if possible. Our minimal-change approach does not impose any changes or add-ons to the core ORB or its interfaces, while facilitating component-based high-precision RTdC.
- *Independence from ORB implementations:* Each ORB vendor is free to choose how to meet the CORBA specifications by its implementation as long as the implementation supports the specified functionality and interfaces. However, experience has shown that the implementations can be very diverse. For example, one group of ORB vendors may implement the core ORB as a set of libraries, while another may create a separate autonomous execution entity for the core ORB. Moreover, each vendor may provide additional non-standard capabilities for the sake of providing convenience to programmers (e.g., bind functions provided by Orbix and VisiBroker, or filter capability by Orbix). Using these capabilities for tailoring CORBA for RT applications (as done in our interception approach mentioned in Section 3) tightly couples the solution to a specific implementation. The minimal-extension approach does not use any non-standard features of current ORB products and, as will be discussed later in this section, the new CORBA services implemented can be adapted with minimal effort when the adopted ORB implementation changes from one to another.

- *Compartmentalization of responsibilities:* In order to build an environment for implementing RTdC applications, several capabilities should be provided by various subsystems. For example, an inter-object communication facility yielding a bounded communication delay should be available for the application. There should be a way for CORBA programmers to express the communication requirements of their applications. As will be discussed later in this section, under the proposed minimal approach, a specific service is devoted to (i) accepting the application communication needs and (ii) cooperating with the underlying operating system and communication subsystems for guaranteeing that these needs will be satisfied.

In the rest of this section, we will discuss the basics of the proposed CORBA-service approach for CORBA-compliant TMO programming.

As depicted in Figure 3, an essential component under the minimal-extension approach is the CORBA service named *TMO Execution Support (TMOES)*. TMOES is the management and scheduling component of the CORBA-compliant TMO network structured system. More specifically, TMOES receives the attributes of each TMO and its associated methods from the application via method-registration steps. Based on the registered information, TMOES manages the timely execution of each registered method. In other words, TMOES is responsible for handling capabilities A, C, and D. Following the principle of responsibility compartmentalization, the communication subsystem of every node should be capable of controlling the part of the communication network under its jurisdiction such that a certain level of quality-of-service (QoS) is guaranteed for communication between itself and each of its neighbor nodes.

Ideally, only the nodes with such capability should be used in hard-RTdC. Assuming that every node possesses such capability, another CORBA service named the *Cooperating Network Configuration Management (CNCM)* can be used to form a group of nodes which have made proper commitments to provide communication channels of desired qualities between themselves and other member nodes. CNCM allows the applications to specify their communication needs and in cooperation with the member nodes hosting various parts (TMO's) of the application, maintains a group of nodes that can meet the communication needs.

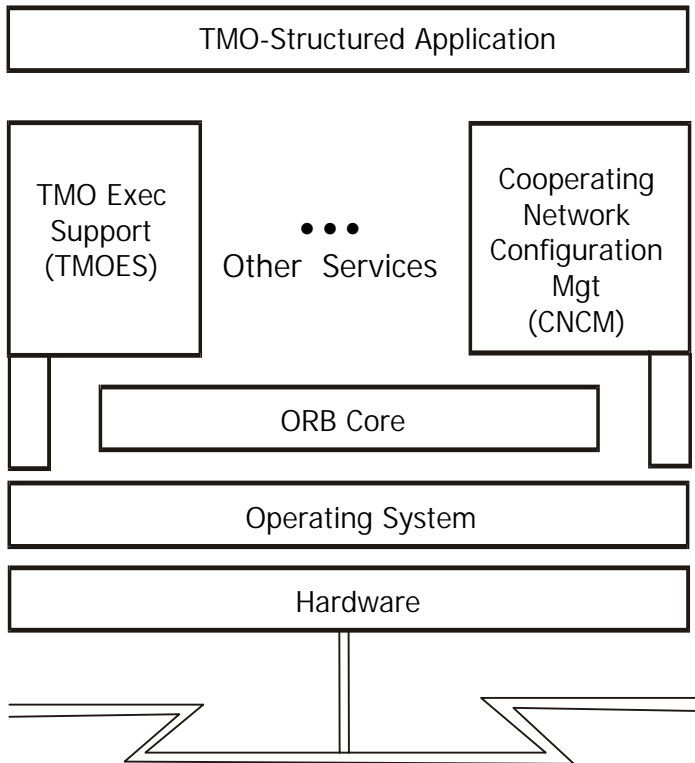


Figure 3. Two CORBA services implemented as middleware supporting TMO networks

#### 4.1 TMO Execution Support (TMOES) as a CORBA Service

TMOES performs the following functions among others.

(1) Registration of timing requirements

A CORBA-compliant TMO uses a standard ORB for remote SvM calls but must register its timing requirements with TMOES. The registered timing requirements include the time-windows for activation of SpM's, the time-windows for completion of both SpM's and SvM's, and the time-windows for completion of I/O activities. Concurrently accessible object data store segments (ODSS's) along with the needs for each method to access certain ODSS's are also registered with TMOES.

(2) Scheduling services

TMOES obtains from the local operating system maximum authority for controlling the local machine resources. Typically this means that TMOES is the highest-priority process in the node and it is rarely interrupted by the local operating system and the duration of each interruption is bounded tightly. Using

the resource controlling authority, TMOES schedules executions of registered methods including the enforcement of BCC (basic concurrency constraint) and multiplexing machine resources among active methods. It also handles automated locking and unlocking of registered ODSS's.

(3) Recognition of calls to local TMO's

When a TMO issues a call for an SvM in another TMO, TMOES can determine whether the latter TMO is local, i.e., resident in the same host node, or in another node. If it is local, then a call for an ORB service can be avoided.

(4) Detection of faults, especially timing faults

TMOES checks whether various components of a TMO are executed within the registered time-windows. When it detects a violation, it can invoke the service of a fault tolerance manager if available.

In our RTdC approach, deadlines are imposed at two different levels for a client-server interaction. A server TMO, to be more specific, an SvM, is associated with a *service time guarantee* which may also be called an *internal deadline*. See Figure 4.

A violation of this internal deadline is detected by the execution engine for the server TMO. If this internal deadline is violated, we view it as an occurrence of a "fault in the server". If the server is equipped with a *fault-tolerant execution manager*, a recovery action may follow.

On the other hand, a client TMO calling for a service from a server TMO imposes a deadline for the *service result return*. This deadline may be viewed as "an *external deadline for the system* supporting the RTdC application". A violation of this deadline is or course detected by the execution engine for the client TMO.

This external deadline will be met when (1) the remote SvM is executed within its internal deadline and (2) the communication of relevant messages between the client and the server is completed within the time budget remaining after subtracting the server's execution time for the SvM from the external deadline. In a nice environment, the client designer can learn at design time that the supplier of the communication network has guaranteed that the amount of message communication in (2) above can always be done within the time budget which is less than the difference between the external deadline to be imposed (by the client designer) and the guaranteed service time (for SvM) advertised by the server TMO implementor. In such a case, the client

designer can in turn issue a certain service time guarantee to its own clients.

If the external deadline imposed by a client is violated for some reasons, e.g., the server failure, the communication network failure, etc., then the client must invoke an exception handler if available. If such an exception handler is not available, then the client also becomes faulty, in which case a recovery action follows only if a fault-tolerant execution manager is available on the client side. TMOSM enforces these two levels of deadlines.

Again, this CORBA service, TMOES, can be implemented as distributed computing middleware that use standard ORB's. Based on our previous experiences [Sho98, Kim99], TMOSM prototypes realizing TMOES with the use of standard ORB's are under development in our laboratory.

#### 4.2 Real-time Cooperating Network Configuration Management (CNCM) through a CORBA service

The CORBA service, Cooperating Network Configuration Management (CNCM), performs the following functions among others.

Suppose an RTdC application is built on the basis of several existing server components of which the locations may or may not be known. The newly developed components of this new RTdC application may be loaded on a new node or new group of nodes. For the time being, let us assume that one new node will host the newly developed components. Among these components is a software component that is responsible for initialization of the new RTdC application and may be called the *RTdC initiator*. This RTdC initiator is aware of the capability of the local node for providing RT communication channels of certain quality between itself and its neighbor nodes. It may have learned about the capability from the operating system of the host node. Therefore, the first step that the RTdC initiator takes is to report to CNCM of the communication capability of the local node, especially the guaranteed quality of the RT communication channels between the local node and the neighbor nodes.

Thereafter, the RTdC initiator informs CNCM of the set of SvM's in pre-existing server TMO's that run on possibly unknown hosts and need to be accessed during its RTdC execution, along with the required quality of the communication channels leading to each remote

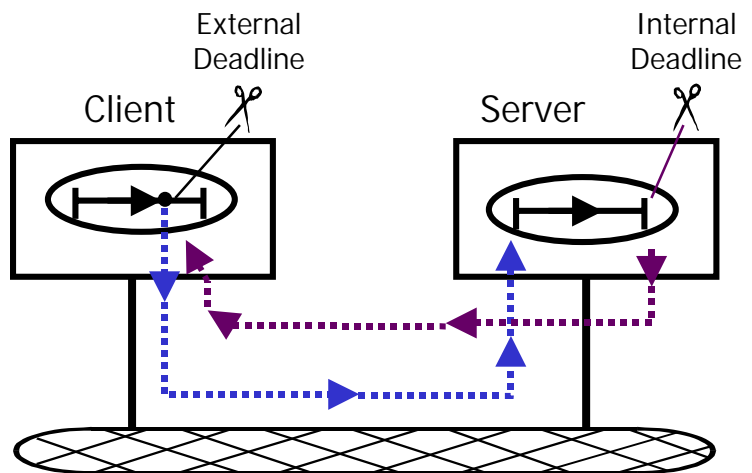


Figure 4. Two levels of deadlines

TMO. For example, the RTdC may say, "a call for SvM X.7 in TMO X from this node must reach the destination in less than 10 milliseconds". Then CNCM replies in one of the following ways.

- (1) "Ok, all TMO's have been located and access channels for them have been found to be of the quality required by you. I have made reservations for your use of those channels. "
- (2) "The access channel for SvM X.7 in TMO X that you plan to use is of the quality below that required by you. Let me know if you want to proceed or not. "

In this case, the RTdC can start and there should be no problem unless component failures occur.

Now one option for the RTdC initiator is to start the RTdC with the understanding that a timing failure may occur. Another option is for the RTdC initiator to cancel the RTdC. The user should then prepare or search for a proper server TMO X by either

- (2.1) having some authority to relocate X to a node which has communication channels of better quality or
- (2.2) having some authority to free some communication channels on the current node hosting X that have been used by another server TMO resident on the same node.

In some situations, CNCM may be able to automatically reconfigure the server TMO X.

From the above discussion, it is straightforward to envision how the restrictive assumption on the newly developed software components being hosted on a single node can be relaxed.

Note that this CNCM approach can be adapted to various environments where different low-level communication protocols are employed. Only the quality

of the communication channels that can be established will change when the low-level protocols change. However, it is always necessary for the communication subsystem in every participating node to be aware of the quality of the communication channels between the node and neighbor nodes. CNCM should collect such information from all members of a group of RTdC nodes and be able to determine the quality of the communication path from one member to another.

## 5. Conclusions

In this paper, we briefly discussed a cost-effective approach for facilitating CORBA-compliant TMO-structured RT distributed application programming. This approach is called the minimal-extension approach because it does not require any changes in the core ORB and IDL. It provides the additional capabilities for executing and connecting TMO-structured CORBA-compliant components via two CORBA services, TMO Execution Support (TMOES) and Cooperating Network Configuration Management (CNCM). Both services can be implemented in the form of distributed computing middleware consisting of CORBA-compliant objects. The initiation and scheduling of the execution of TMO methods are managed by TMOES and its middleware implementation is called the TMO support middleware (TMOSM). We have implemented a TMOSM on the Windows NT platform equipped with OmniORB2 and it is under refinement to fully conform to the CORBA service structure. A prototype implementation of CNCM will follow thereafter.

**Acknowledgments:** The research work reported here was supported in part by the US Defense Advanced Research Project Agency under Contract N66001-97-C-8516 monitored by SPAWAR, and in part by Hitachi, Ltd.

## References

- [Att91] Attoui, A. "An Object Oriented Model for Parallel and Reactive Systems", *Proc. IEEE CS 12th Real-Time Systems Symp.*, 1991, pp. 84-93.
- [Ish92] Ishikawa, Y., Tokuda, H., and Mercer, C. W., "An Object-Oriented Real-Time Programming Language", *IEEE Computer*, October 1992, pp. 66-73.
- [Kim94] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS '94)*, Oct. 94, Dana Point, pp.36-45.
- [Kim97] Kim, K. H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No. 8, August 1997, pp. 62-70.
- [Kim99] Kim, K. H., Ishida, M., and Liu, J., "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", to appear in *Proc. 2nd IEEE CS Int'l Symp. on Real-time Object-oriented distributed Computing (ISORC '99)*, May 1999, St. Malo, France.
- [OMG95] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, 1995.
- [Sho97] Shokri, E., et al., "An Approach for Adaptive Fault Tolerance in Object-Oriented Open Distributed Systems", *Proc. 1997 IEEE CS Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '97)*, Newport Beach, CA, February 1997, pp. 298-305.
- [Sho98] Shokri, E., Crane, P., and Kim, K.H., "An Implementation Model for Time-Triggered Message-Triggered Object Support Mechanisms in CORBA-Compliant COTS platforms", *Proc. 1st IEEE CS Int'l Symp. on Real-time Object-oriented distributed Computing (ISORC '98)*, April 1998, Kyoto, Japan, pp. 12-21.
- [Tak92] Takashio, K., and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", *Proc. OOPSLA*, 1992, pp. 276-294.

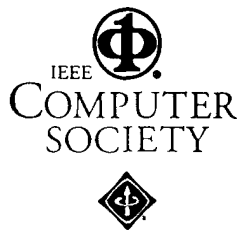
Proceedings

# Fourth International Workshop on Object-Oriented Real-Time Dependable Systems

January 27-29, 1999  
Santa Barbara, California, USA

*Sponsored by*

IEEE Computer Society



Los Alamitos, California

Washington • Brussels • Tokyo

---