

# Toward Globally Optimal Resource Management in Large-Scale Real-Time Distributed Computer Systems

K. H. (Kane) Kim  
Department of Electrical & Computer Engineering  
University of California  
Irvine, CA 92697, U.S.A.  
kane@ece.uci.edu

**Abstract:** This paper discusses the issues and promising approaches in (1) obtaining rigorous specifications of the quality-of-service (QoS) requirements associated with application functions and (2) using such specifications as the main driver for cost-effective resource allocation. The application environments in which component failures must be handled with sufficient efficiency in order not to miss the application objectives and also execution resource requirements of various application functions can change dynamically, are considered. The importance of structuring various resource users, i.e., application software components and middleware components, in easily analyzable forms, is emphasized. A promising middleware architecture that supports application software structured in real-time object-oriented forms and also manages execution resources for fault-tolerant system operations, is presented.

**Index Terms:** resource allocation, scheduling, quality of service, risk, risk incursion function, potential risk, middleware, fault tolerance, real time.

## 1. Introduction

Nowadays complex real-time computer systems (RTCS's) are invariably of the distributed system type. In such systems many multi-purpose modules share the workload for providing a variety of time-critical service functions to achieve application objectives. Local area network (LAN) or wide area network (WAN) communication facilities are also shared in message exchanges among various nodes and tasks handling the time-critical service functions. The mapping of various computation segments to these execution machine components have been treated as a major research and development issue for at least two decades [Son94].

In general, the resource allocation in large-scale systems should involve activities at multiple levels such as the WAN level, the LAN level, the node level, and the processor level. Such resource allocation cannot be adequately handled by straightforward extensions of the CPU and peripheral device scheduling techniques used in non-RTCS's. A fundamental limitation common in most established approaches is that they are based on the use of *excessively simplistic characterizations of computation-segments competing for use of the execution resources.*

For example, there has been growing recognition that assigning fixed priorities to low-level resource users such as processes or tasks is a very crude way of estimating the resource needs of application functions.

The purpose of this paper is to discuss the issues and promising approaches in (1) obtaining *rigorous specifications of the quality-of-service (QoS) requirements* associated with application functions and (2) using such specifications as *the main driver for cost-effective resource allocation.* The application environments in which component failures must be handled with sufficient efficiency in order not to miss the application objectives and also execution resource requirements of various application functions can change dynamically, are considered. In such environments, effective resource allocation leads directly to maximizing the survival periods of the application systems. Therefore, resource allocation in manners adaptive to component failures and application mode changes is one of the major issues discussed in this paper.

The paper starts with the discussion on the needs for system-level *QoS-driven resource allocation* in Section 2. A promising approach for QoS requirements specification is also discussed.

Then in Section 3, the importance of structuring various *resource users, i.e., application software components and middleware components,* in easily analyzable forms, is emphasized. A recently developed powerful real-time object structuring approach which is effective in structuring easily analyzable real-time application objects [Kim94b, Kim97b], is used to illustrate such a desirable form of structuring.

The subject of structuring middleware components in easily analyzable forms is discussed in Section 4. A promising middleware architecture that supports real-time fault-tolerant objects is introduced as an example.

The paper is concluded in Section 5.

## 2. Quality-of-service (QoS) requirement specification as the driver for resource allocation

### 2.1 Problems with conventional resource allocation approaches based on crude estimation of the resource requirements

Old crude ways of estimating the resource needs of application functions such as assigning fixed priorities to

processes, were acceptable in simple centralized real-time control systems. For example, they were acceptable in a simple device control system in which the main processor runs multiple fixed-pattern periodic processes, each repeatedly exchanging signals/messages with one specifically assigned cyclic device in a fixed pattern without much interaction with other processes.

However, they are too crude to be effective in large-scale distributed RTCS's running many cooperative dynamic decision-making application subsystems which exhibit dynamically changing patterns of interaction among themselves as well as with the application environments. *Resource users and resource needs must be considered in a top-down fashion* as elaborated below.

Ultimately, execution resource requirements come from the needs to produce acceptable-quality outputs of application functions. The most meaningful purpose of any non-random resource allocation is to meet the application requirements *with the best quality of execution results and with minimal use of execution resources*. Therefore, whatever resource allocation techniques are adopted, their effectiveness must be evaluated on the basis of how well *the output quality requirements, i.e., quality-of-service (QoS) requirements of application functions* are met after using them.

An output of an application function may be the result of complex interactions among many software components running on distributed nodes. Therefore, distribution of application computation-segments among WAN nodes (each of which may be a LAN) and possibly further distribution among LAN nodes are important resource allocation activities that must occur before focusing on the approaches for resolving the competitions among the co-resident computation-segments within each node for using intra-node execution resources. This *hierarchical resource allocation* must be done systematically and the state-of-the-art in this is currently very primitive. The QoS requirements for the types of application functions considered here cannot be adequately specified in terms of fixed priorities (attached to a fixed set of processes) or even in terms of completion deadlines only.

Moreover, QoS attributes that are of interest to the system engineer or customer may include not only the timeliness and the accuracy of the application function outputs that are achieved during peaceful fault-free operations but also those achievable *under occasional occurrences of component failures* and/or the *ability to thwart security threats* [Law95]. None of these QoS attributes can be properly translated into fixed priorities attached to a fixed set of processes and most of them cannot be adequately represented by computation-segment completion deadlines only.

In hierarchical resource allocation, the higher-level resource allocations, i.e., WAN-level and LAN-level

distributions of computation-segments, are generally more important in terms of their impacts on the system-wide cost-effectiveness of resource allocation than the lower-level allocation of intra-node resources.

In short, with conventional resource allocation approaches, QoS requirements of application functions in large-scale distributed RTCS's cannot be met in cost-effective manners. Such approaches more often than not result in poor QoS's of application functions compared to the QoS's that can be obtained under more intelligent resource allocations.

Significant advances in reliable and cost-effective design of large-scale distributed RTCS's can thus be achieved only when good understanding is obtained on the relationship between various types of QoS requirements and various feasible hierarchical resource allocations.

## 2.2 QoS attributes and causes for QoS losses

The typical service actions of an RTCS are combinations of

- (a) outputting control values to devices in the application environments and
- (b) storing newly computed values into a database which is shared by users outside the control domain of the RTCS.

An RTCS is required to take every service action accurately not only in "time dimension" but also in "logical value dimension" [Kim96]. The value of an output of an RTCS is thus a two-dimensional value called a *timed logical value*. The accuracy or timed value accuracy of an output of an RTCS is the closeness of both time and logical value attributes of the output to the time and logical value attributes of the desired output. Similarly, the temporal accuracy of any data transmission or storing action in an RTCS is the closeness of the time attribute of the action to the time attribute of the desired action, while the logical value accuracy of the action is the closeness of the logical value attribute of the data involved in the action to the logical value attribute of the desired action. Therefore, the timed value accuracy requirements imposed on each output are the core requirements of the system, which are also called the *functional requirements*.

Examples of the manifestations of inaccurate output observed often in practice are:

- (1) *Omission* of a desired output, i.e., output with zero degree of temporal accuracy;
- (2) *Too late output*, i.e., output with an unacceptably low degree of temporal accuracy;
- (3) *Unacceptable logical value*, i.e., output with an unacceptably low degree of logical value accuracy.

In considering why we have to deal with these undesirable events and how to avoid or deal with them, it is good to distinguish the following two types of situations:

(Env1) There are no fault occurrences and no security threats.

(Env2) Faults and/or security threats may occur during system operation.

In application environments of Env1 type, inaccurate outputs can occur only as a result of one of the following: (Excuse1) *careless design*, (Excuse2) *unanticipated workload*, or (Excuse3) use of *unpredictably time-consuming components*.

The cause Excuse1 should be avoided in safety-critical application fields although it may never go away completely due to human limitations. In any case, it is outside the scope of the discussion in this paper.

The cause Excuse2 should also be avoided at least in important parts, if not in the entirety, of large-scale RTCS applications. Not knowing the peak loads to be present in important parts of the systems is no different from careless design. As the costs of computer hardware no longer represent major parts of the costs of the overall application systems in most large-scale application environments, letting inaccurate outputs occur merely because of miscalculations of the possible workloads or interests in saving hardware costs is no longer justified.

In principle, the cause Excuse3 is not easily justified, either. Computer hardware contributes to unpredictable timing behaviors of software but given the current states in the hardware economy, it is difficult to argue for using hardware components which make big contributions to unpredictable timings in safety-critical RTCS's. However, operating system contributions to unpredictable timings has been a different situation. It has been an issue of major concerns to RTCS designers in the past decade. As a result, substantial improvements have been achieved in this technology area. At least the principles for constructing timeliness-guaranteed local operating systems suitable for Env1-type application environments are now well understood. Quite a few commercial local operating systems with reasonable degrees of timing predictability that can be used in such application environments are now available. A more serious problem is now with the much less predictable timings of currently available distributed system management middleware services. This situation is expected to improve rapidly in the next decade.

On the other hand, in application environments of Env2 type, it is an order-of-magnitude more difficult, if not impossible, to avoid inaccurate output actions. Some messages in transmission may be lost due to faults in some hardware components involved. Or as some processors are lost due to faults, the processing power may become insufficient to produce all outputs in time. As execution resources dwindle down below a certain level, some application functions will be sacrificed entirely or the new allocation of resources to them will result in lowering of their QoS's. Such resource

reallocation must be based on clear understanding of the QoS impacts of the reallocation. Such understanding can never be obtained if fault tolerance management middleware components are not *easily analyzable*.

Fault tolerance requirements must be an integral part of the initial system QoS requirements. Selection of appropriate system designs to meet the fault tolerance requirements involves both static allocation of some execution resources and design of some dynamic allocation algorithms. The static allocation is often a part of the high-level design of the system, e.g., WAN-level redundancy design. Advances in design to meet fault tolerance requirements will have much bigger impact on the cost-effectiveness of RTCS's than further improvement of conventional low-level process schedulers operating with little information on system-wide resource requirement patterns.

Therefore, in Env2-type application environments, there are three major causes for run-time resource allocation actions at a level higher than the finest time-grain processor scheduling level as shown in Figure 1:

- (1) fault detection,
- (2) repair and reincorporation of previously unusable components, and
- (3) entering of the application into a new phase where resource requirements are significantly different from the preceding phase.

### 2.3 Risk incursion function (RIF) based QoS requirement specification

As mentioned in preceding sections, important technological advances in resource allocation can occur only if top-level QoS requirements can be clearly understood and specified and methods for allocating resources to meet the QoS requirements optimally become available. Given a set of service functions defined as the mission of an RTCS under development, the basic required resource set can be defined as a minimum set of resources which collectively provide enough processing capacity to support the mission when the possibility of any component failure during the application life-time can be ruled out. Therefore, mobilizing any additional resources beyond the basic required resource set will return no benefits as long as no components in the basic set fail. However, component failure probabilities are non-negligible in practice and thus some extra resources not included in the basic required resource set must be incorporated into the RTCS. In fact, as mentioned before, component failures are the major cause for necessitating difficult run-time resource allocation actions and eventual QoS losses.

System engineers must endeavor to understand not only the QoS requirements (i.e., the output accuracy, fault tolerance, and security requirements) but also the impacts of QoS losses, i.e., inaccurate outputs on the overall

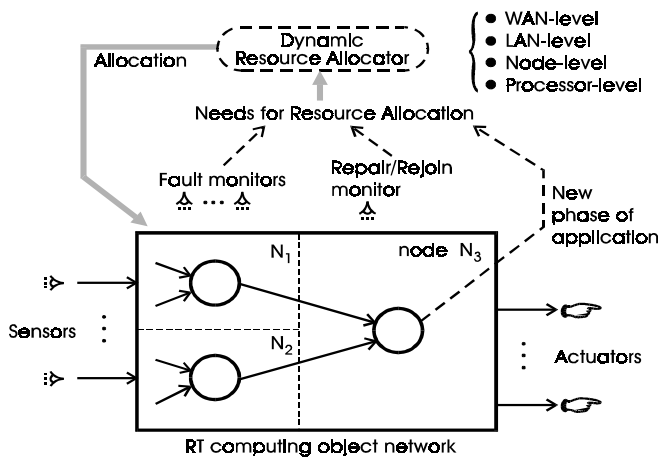


Figure 1. Needs for run-time resource allocation (Adapted from [Kim96])

application success. The potential damaging impacts may be called risks (also called benefit losses [Kim96]). Whenever resources are tight and competitions arise, allocation decisions must be made in the direction toward minimizing the risks.

The quantitative specification of the relationship between the loss in the timed value accuracy of each output action (due to faults, resource shortage, etc.) and the consequent damage to the application mission is called the risk incursion function (RIF) (also called benefit loss function) of that output. The RIF can be viewed as a refinement of the time-value function proposed earlier by Jensen [Jen85]. In short,

$$\begin{aligned} \text{RIF} &:= \text{relation}(\text{Loss in the timed value accuracy} \\ &\quad \text{of each output action, Application damage}) \\ &:= \text{relation}(\text{QoS loss, Risk}) \end{aligned}$$

An RIF-based QoS requirement specification must include the following:

- (RS1) an RIF;
- (RS2) a specification on the types of fault occurrences that should not be ignored;
- (RS3) a specification on the types of security threats that should not be ignored;
- (RS4) a specification of other constraints such as cost constraints, power consumption constraints, and physical volume constraints, etc.

To be more specific, when the accuracy loss  $\Delta_v$  in an output action  $v$  occurs, the application risk  $r(\Delta_v)$  is incurred. The RIF is thus the mapping of every accuracy loss event to a consequent application risk value. The determination of  $r(\Delta_v)$  for every output action  $v$  in an RTCS is the responsibility of the system engineer. When the application risks accumulated over a certain period of time exceed a certain predetermined threshold, the RTCS can be treated as having failed in its application mission. The threshold can thus be called a system failure threshold and it should be determined by the system engineer.

The specification component RS2 is typically given in terms of the maximum occurrence frequency and/or maximum interval between occurrences for each type of faults. A fault type is typically characterized by the type of the component exhibiting the fault and the duration of the fault, e.g., permanent vs. transient.

An RIF-based QoS requirement specification is a solid baseline driver in globally optimal resource allocation.

### 3. Structuring of easily analyzable resource users

Given a rigorous RIF-based specification of the quality-of-service (QoS) requirements, the system engineer must produce a system design capable of meeting the QoS requirements with minimal use of execution resources. Each system design reflects some static allocation decisions and contains some run-time resource allocation procedures. Therefore, *meeting QoS requirements in an optimal way is essentially an optimal system design activity.*

Each run-time resource allocation procedure is designed to resolve dynamically occurring competitions among various computation-segments for using a certain execution resource. Each run-time resource allocation must be made toward the ultimate goal of minimizing the risks incurred while making minimal uses of execution resources. To be practical, a run-time resource allocation procedure must perform only a small amount of computation in analyzing the characteristics of every competing resource request and determining the use order. Therefore, the *characteristic description* accompanying each resource request presented to a run-time resource allocation procedure must be compact and must have been derived from the original system-level QoS requirement specification.

Such a characteristic description is essentially a description of the potential impacts of the resource use by the requesting computation-segment on the system-level QoS. Finding a good characteristic description is the most challenging task for the system engineer (team) after the task of producing an RIF-based QoS requirement specification. Later in this section, an approach called the risk incursion potential function (RIPF) approach will be discussed.

In generating the characteristic descriptions of the resource requests of various computation-segments by reflecting the QoS requirements, an easily understandable and yet rigorous representation of the system being developed and its application environment is of innumerable value to the system engineer. One such representation approach is the TMO (time-triggered message-triggered object, also called RTO.k) structuring scheme, a recently formulated major extension of the conventional object structuring [Kim94b, Kim97b]. In the rest of this paper, we will assume without loss of

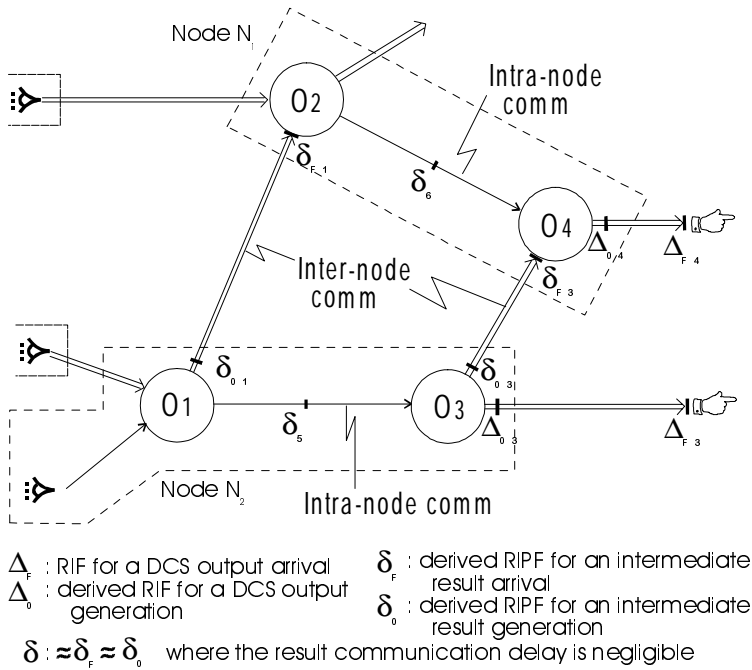


Figure 2. System-level RIF and derived RIF (adapted from [Kim96])

generality that RTCS's are designed in the forms of TMO networks to make the discussions on resource allocation issues in unambiguous forms.

The TMO structuring scheme provides the *uniformity* and the *wide range of controlled accuracy* in the representation of both complex RTCS designs and their application environment descriptors/simulators that evolve during the system development cycle. The scheme supports *general-form* (as opposed to esoteric) design of RTCS's. The scheme is also aimed for drastically reducing the designer's efforts in producing highly reliable complex RTCS's, especially in producing RTCS's with *timely service guarantees*. The abilities of the scheme to represent distributed computing resources and application environments in accurate and easy-to-analyze manners facilitate modular and rigorous specification of functional requirements. The TMO structured functional requirement specification is a natural framework within which QoS requirement specifications can be incorporated.

The issue of generating the characteristic descriptions of resource requests by various computation-segments can now be discussed in the context of TMO-structured RTCS's. In RTCS's structured as TMO networks, each subsystem can be viewed as a computation TMO or *computation object* (C-object) as illustrated in Figure 2. Some outputs from C-objects are *system outputs* going to devices in application environments or to the database outside the control domain of the RTCS. RIF's are associated with these system output actions. Other

outputs from C-objects go to other C-objects. Accuracy loss at a C-object may thus contribute to loss in accuracy of system outputs.

Initially, an RTCS can be represented as a single C-object O which interacts with the external environment by accepting inputs from the sensors and producing outputs to the actuators. As a part of producing the QoS requirement specification, the system engineer determines the RIF( $\Delta_f$ ) for each action of a system output reaching an actuator in the environment, where  $\Delta_f$  is the accuracy loss occurred in a system output *reaching* the actuator. In the system where the C-object and the actuator are located in the same node (i.e., the communication delay between the C-object and the actuators is negligible), we can assume  $\Delta_f = \Delta_0$  where  $\Delta_0$  is the accuracy loss occurred in a system output *produced*.

The design process decomposes the RTCS initially represented as a single C-object into a set of several C-objects, say  $O = \{O_1, O_2, \dots, O_n\}$ , which interact with one another as illustrated in Figure 2. Each C-object produces outputs going to other C-objects or the environment. Outputs going to other C-objects are called *intermediate outputs*.

One reasonable approach for deriving cost-effective characteristic descriptions of resource requests from an RIF-based QoS requirement specification is to produce for each intermediate output a *risk incursion potential function* RIFP ( $\delta_f$ ), where  $\delta_f$  is the accuracy loss occurred in the intermediate output reaching another C-object. RIFP is essentially the following relation.

$$\text{RIFP} := \text{relation}(\text{Accuracy loss in intermediate output, Potential risk})$$

Producing an RIFP for an intermediate output involves the determination of:

- (1) an appropriate deadline (or a timed value accuracy requirement) for the intermediate output reaching the receiver C-object and
- (2) the realistic possibility of an untimely (e.g., too late) (or inaccurate) intermediate output leading to untimely (or inaccurate) system outputs.

In principle, the *potential risk* that can be incurred by an untimely execution of the intermediate output function  $w$  is

$$\sum_{u_k \in U} \{\text{risk that can occur through an untimely or}$$

inaccurate action of system output  $u_k$  caused by that untimely execution of  $w\}$ ,

where  $U$  is the set of all system output functions.

Therefore, an RIFP derived in strict adherence to this principle will become more complex than the system-level RIF's. In practice, this principle will be followed only approximately by the system engineer who is interested in producing simple and easy-to-use RIFP's.

The system engineer must produce RIPF's conservatively so that intermediate output actions producing no potential risks can always lead to accurate system outputs in the absence of any other anomalies.

Since deriving RIPF's from an RIF based QoS requirement specification is not a mechanical process, various analysis tools that can aid the system engineer in this task are important future research subjects.

The decomposition process will continue until the C-objects become the basic units for resource allocation. In this course of producing multiple levels of C-objects, RIPF's for the output functions of each C-object must be produced. So, RIPF's associated with high-level C-objects are used as drivers for generating RIPF's associated with low-level C-objects. In this sense, RIPF's are produced in a top-down hierarchical manner. Concurrently, static allocation of execution resources and design of run-time resource allocation procedures occur in a similar manner.

For scheduling of an intra-node resource such as a CPU, the basic requesting unit is typically a process which is assigned to a method of a TMO in a TMO-structured system. After the RTCS is decomposed into a network of C-objects which are basic competing units for resources, each case of a C-object presenting to a local resource allocator its estimates for worst-case execution times before its output actions along with the RIPF's associated with its output actions, is analyzed to ensure that the local resource scheduler is never over-taxed under fault-free circumstances. It should also be noted that whenever  $\Delta_F$  and  $\Delta_O$  in Figure 2 (similarly,  $\delta_F$  and  $\delta_O$ ) are different, then they are used by the communication network scheduler in scheduling the transmission of each output value generated.

Not surprisingly, a simple experimental study conducted recently has indicated that RIPF-driven local resource allocators will allocate resources more effectively toward global optimality (i.e., minimizing risks) than existing allocators that utilize fixed priority and simplistic deadline information only [Kim96]. It is also clear that a TMO-structured functional requirement specification serves as a natural framework within which an RIF-based QoS requirement specification can be incorporated. As the integrated specification is refined in a top-down hierarchical manner, RIPF's are also generated and refined in a similar manner.

#### **4. An architecture for middleware supporting adaptable real-time fault-tolerant objects**

The importance of structuring one of the two major classes of resource users, application software components, in easily analyzable forms, was emphasized in the preceding section. Structuring the other major class of resource users, middleware components, in easily analyzable forms is no less important.

A promising middleware architecture that supports application software structured in real-time object-oriented forms mentioned in the preceding section, i.e., TMO-structured application software, and also manages execution resources for fault-tolerant system operations, is discussed in this section. The architecture yields a relatively easy analysis of the temporal and logical behavior of various middleware services. It facilitates handling *in tightly bounded time* the failures of computing and communication components of both hardware and software types. Figure 3 depicts the middleware architecture.

The middleware components run on a kernel. Four middleware components are structured as periodic system-threads. They receive fixed-size time-slices periodically and thus their actual execution times are quite obvious. Their service functions are as follows:

##### *(1) TMO support manager (TMOSM)*

The TMOSM provides execution support for TMO's, especially, CORBA-compliant TMO's. CORBA is a vendor-independent standard which aims at interoperability and portability of distributed object-oriented applications [OMG95]. This standard provides a high-level location-transparent inter-object communication and thereby makes the development of distributed applications simpler. The TMOSM includes the following two major components:

- *RIPF-driven processor scheduler*: The RIPF-driven scheduler schedules the real-time processes towards the objective of minimizing the total potential risks, where the processes dynamically present their worst-case computation time estimates, deadlines, and RIPF's associated with their next computation-segments to be executed.

- *RIPF-driven communication resource (LAN channels and WAN tickets) allocator*: The RIPF-driven communication resource allocator allocates the LAN channels or WAN tickets to the real-time messages toward the objective of minimizing the total potential risks, where the envelope of each message presents the size, the deadline, and the RIPF associated with the message transmission.

##### *(2) Data field (DF) manager*

The DF manager facilitates creation of logical multicast channels called DF channels [Mor93, Kim95] over physical broadcast LAN channels, connection of processes to the DF channels, and delivery of messages to relevant channel users. It enables processes and TMO's to communicate among themselves in location-transparent manners. The DF channel is a more general mechanism than the port mechanism in UNIX environments in the sense that the former is a multiple-sender multiple-receiver facility whereas the latter is a multiple-sender single-receiver facility.

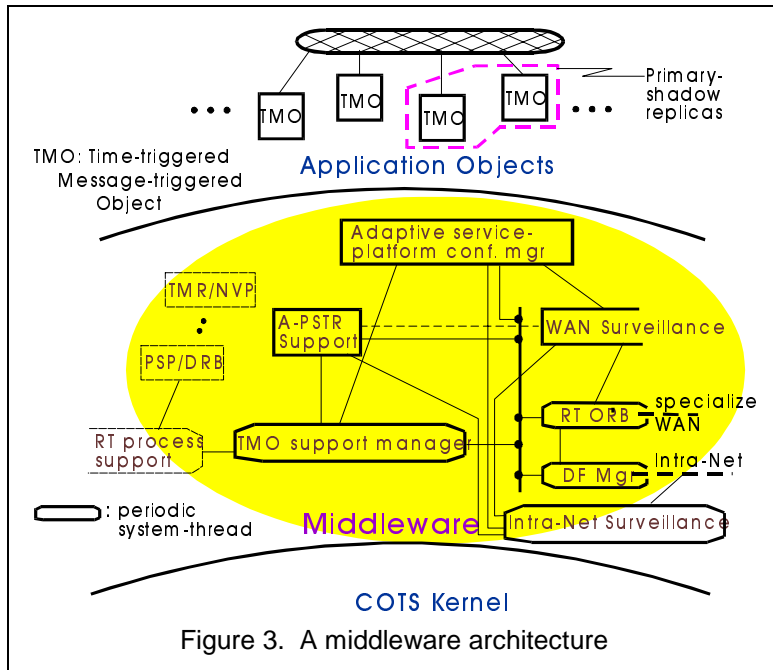


Figure 3. A middleware architecture

### (3) Real-time object request broker (ORB)

Current CORBA-compliant ORB's are not suitable for use in real-time applications mainly because the CORBA standard does not provide mechanisms for specifying timeliness properties such as deadlines for message delivery and method execution. However, a real-time ORB which supports the following operations is expected to become available in the near future.

- (a) The client object is able to impose a deadline for a message to arrive at the server object(s).
- (b) The client object is able to impose a deadline for the server object(s) to complete a requested service method execution and deliver the result back to the client.

Therefore, the real-time ORB manager must operate in a way that the specified deadlines are met by using available communication and computation resources. The ORB may use both WAN channels and LAN channels since server objects may be resident in near LAN nodes as well as in remote WAN nodes. Use of LAN channels is realized by receiving the cooperation of the DF manager.

### (4) Intra-net surveillance

Network surveillance is essential in order to enable each interested fault-free node in the system to learn fast of the faults or repair completion events occurring in the other parts of the system [Kim94a]. Network surveillance can thus handle the most critical part of the fault detection requirements in the system. Surveillance of LAN channels is handled by this middleware component. Sound schemes for real-time LAN surveillance have become available from recent research efforts [Kop93, Kim94a, Kim97c]. The key common property of these schemes is the *tightly bounded detection latency*.

Other middleware components in Figure 3 need not be structured as periodic system threads since their execution frequencies need not be as high as those for the four components described above. They can be implemented as real-time processes that are less frequently executed and may be executed within much larger time windows than the windows within which the execution start times of system threads can be chosen. Their service functions are as follows.

### (5) WAN surveillance

The scope of the WAN surveillance is broader than that of the intra-net surveillance. The former needs not be as frequent as the latter. Even so, the detection latency of this middleware component must still be tightly bounded although the granularity of the time unit can be larger than that in the intra-net case. When it performs a surveillance of WAN channels, it receives the cooperation of the real-time ORB. It also receives periodic reports from the component handling intra-net surveillance.

### (6) Adaptable primary-shadow TMO replication (PSTR) support and other components supporting fault-tolerant execution of real-time processes

Widely applicable adaptable real-time fault tolerance schemes possessing small recovery time bounds have been formulated and proven to reasonable extents, most of them in the past 20 years. The fundamental types of widely applicable schemes include the TMR/NVP (triple modular redundancy / N-version programming) scheme [Avi95], the DRB/PSP (distributed recovery block / pair of self-checking processing nodes) scheme [Kim94a], and the PSTR (primary-shadow TMO replication) scheme which is a time-bounded fault tolerance scheme for realizing real-time object replicas [Kim97a], etc. Adaptable versions of these schemes, which enable dynamic changes in the degree and type of redundancy as resource availability changes, have recently started emerging [Kim92]. Such versions take advantage of the fault/rejoin detection services of the intra-net surveillance and the WAN surveillance components.

### (7) Adaptive service-platform configuration manager (SPCM)

The adaptive SPCM maintains the set of various service-platform configurations of the RTCS's and upon a component failure or rejoin (after repair), selects one by reflecting the RIPP's. Obviously, the RIPP-driven decision is made on the basis of which configuration will result in the minimal total potential risks. The SPCM's reconfiguration is also triggered when the application enters into a new phase where resource requirements are significantly different from the preceding phase.

Once the SPCM decides on a new configuration, the middleware components discussed in (6) above are

responsible for accepting the part of the decision relevant to them and reconfiguring the relevant nodes and computation-segments under its jurisdiction into the newly decided execution mode. In this way, QoS/RIPF - driven management of the service-platform configuration and adaptation of fault tolerance modes and local resource allocators is realized.

In summary, the middleware architecture discussed above has been devised to realize QoS/RIPF - driven resource management in manners adaptive to component failures and application mode changes. Since middleware components themselves are important resource users, an emphasis has been put on making their temporal and logical behavior easily analyzable. There are many implementation aspects which deserve extensive research. One prototyping effort is under way at the author's location.

## 5. Conclusion

In this paper, we advocated the QoS requirement driven globally optimal resource allocation in large-scale distributed RTCS's. As a concrete direction for implementing such a scheme, an RIF-based rigorous approach for QoS requirement specification and an approach for derivation of RIPF's from the original QoS requirement specification and using the RIPF's in run-time resource allocation was discussed.

The importance of structuring both application components and middleware components in easily analyzable forms was emphasized. Specific tools for such structuring, i.e., the TMO structuring and the newly proposed architecture for middleware services for QoS/RIPF-driven resource management in manners adaptive to component failures, were also presented.

The QoS/RIPF-driven adaptive resource management will result in minimizing the application risks which means maximizing the survival periods of the real-time distributed computing application systems.

The subject of *QoS-driven adaptive resource allocation in large-scale RT distributed computing (DC) application systems* is still a young one. In order to establish it as a widely practicable technology, extensive research must be performed in the future on systematic derivation of cost-effective RIPF's from the initial RIF-based QoS requirement specification as well as on formulation and efficient implementation of middleware architectures like the one presented in this paper.

**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 NRD, in part by the University of California's MICRO Program under Grant 96-169, in part by Hitachi, Ltd., and in part by LG Electronics.

## References

- [Avi95] Avizienis, A., "The Methodology of N-Version Programming", Ch. 2 in M.R. Lyu ed., '*Software Fault Tolerance*', John Wiley & Sons, 1995, pp.22-46.
- [Jen85] Jensen, E.D., Locke, C.D., and Tokuda, H., "A Time-Value Driven Scheduling Model for Real-Time Operating Systems", *Proc. IEEE CS Symposium on Real-Time Systems*, Nov. 1985.
- [Kim92] Kim, K.H. and Lawrence, T.F., "Adaptive Fault-tolerance in Complex Real-Time Distributed Computer System Applications", *Computer Communications*, May 1992, Vol.15, No.4, pp.243-251.
- [Kim94a] Kim, K.H., "Action-level Fault Tolerance", Ch. 17 in Sang H. Son ed., '*Advances in Real-Time Systems*', Prentice Hall, 1994, pp. 415-434.
- [Kim94b] 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)*, Oct. 94, Dana Point, pp.36-45.
- [Kim95] Kim, K.H., Mori, K., and Nakanishi, H., "Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HUDF Inter-Process-Group Communication Scheme", *Proc. 1995 IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS)*, April 1995, Phoenix, pp.305-312.
- [Kim96] Kim, K.H. et al., "An Experimental Investigation of the Potential of BLF-driven Scheduling of Real-time Threads", *Proc. 2nd IEEE Int'l Conf. on Engineering of Complex Computer Systems (jointly with CSESAW '96, RTAW '96, & SES '96)*, Montreal, Canada, Oct. 1996. pp.60-67.)
- [Kim97a] Kim, K.H. (Kane) and Subbaraman, C., "Fault-Tolerant Real-Time Objects", *Communications of the ACM*, Vol. 40, No.1, January 1997, pp. 75-82.
- [Kim97b] K.H. Kim, "Object Structures for Real-Time Systems and Simulators", to appear in *Computer (IEEE Computer Society Magazine)*, August 1997.
- [Kim97c] K.H. Kim and C. Subbaraman, "A Supervisor-Based Semi-Centralized Network Surveillance Scheme and the Fault Detection Latency Bound", to appear in *Proc. SRDS 97 (IEEE Computer Society's Symp. on Reliable Distributed Systems)*, Oct. 1997.
- [Kop93] Kopetz, H. and Grunsteidl, G., "TTP-A: Time Triggered Protocol for Fault Tolerant Real-Time Systems", *Proc. IEEE CS FTCS-23*, Toulouse, France, June 1993, pp. 524-533.
- [Law95] Lawrence, T.F., "Anomaly Management in Complex Systems", *Proc. PRFTS '95*, Newport Beach, Dec. '95, pp. 132-134.
- [Mor93] Mori, K., "Autonomous Decentralized Systems: Concept, Data Field Architecture, and Future Trends", *Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 93)*, Mar.93, Kawasaki, Japan, pp. 28-34.
- [OMG95] Object Management Group, "The Common Object Request Broker: Architecture and Specification", Revision 2.0, 1995.
- [Son94] Son, Sang H. ed., '*Advances in Real-Time Systems*', Prentice Hall, 1994