

QoS-driven Resource Management in Real-Time Object Based Distributed Computing Systems

K. H. (Kane) Kim and Juqiang Liu

DREAM Laboratory
University of California
Irvine, CA 92697, U.S.A.
<http://dream.eng.uci.edu>

Abstract: Quality-of-Service (QoS) driven resource management is a promising notion but has remained far from being established as a sound technology of a general form. The first step needed is to develop a practical and rigorous approach for specification of the QoS requirements associated with application functions. Such an approach must be accompanied by an approach for using such specifications as the main driver for cost-effective resource allocation. A promising concept, later named risk incursion function (RIF)-driven resource management, was created in 1994 and since then, slow efforts to convert the concept into a concrete technology have been under way. This research has been following another complimentary research on a promising middleware architecture that supports application software composed of distributed real-time objects and also manages execution resources for fault-tolerant system operations. In this paper, we discuss the current middleware structure into which mechanisms facilitating RIF-driven resource allocation can be incorporated with relative ease, as well as several practical patterns of RIF that have been established recently.

Keywords: Real-time, RIF, Risk Incursion Function, RIPP, Risk Incursion Potential Function, TMO, TMOSM, TMO Support Middleware.

1. Introduction

In spite of the continuing decline of computer hardware costs, allocation of computer resources is still a major issue in designing complex real-time (RT) computer systems such as those needed in transportation automation, defense command and control, telecommunication, etc. In such systems, tight resource conditions can rise due to the failure of computing components. Moreover, the real-time recovery of the computation disturbed by the faulty components also involves resource allocation actions.

In the design of RT computer systems (RTCSs), the industry practice is still largely to assign fixed priorities to processes, especially for the centralized RT control systems. However, assigning fixed priorities is a very *primitive* and *crude* way of expressing the relative *importance* or *urgency* among different tasks or processes. Major weaknesses of such an approach for

use in RT distributed computing (DC) systems are as follows:

(1) Major complexity reduction in design of RT DC systems can be achieved by allowing the system designer to deal with high-level abstract computation units, e.g., distributed objects [Bol00, OMG01, Kim00b], while leaving the details of mapping the high level design to the execution resources to the underlying intelligent execution engines. When assigning fixed priorities, the system designers are exposed to the low-level execution units, such as processes and threads, which should be avoided if possible.

(2) Except in simple cases of centralized systems, the system designer cannot easily map the timing requirements of the concurrently advancing application processes to a fixed priority assignment. In other words, the relationship between the timing requirement and the timing behavior effected by assigning fixed priorities to processes can not be clearly recognized in most cases.

In the application environment considered here, application software is composed of distributed RT objects. If there are timing requirements inherent in the target applications, it should be expressed in the simplest easily analyzable form in the high-level system design. If an application requires one object method to be started within a certain time window and completed within a certain deadline, the starting time-window and the completion deadline should be expressed in a natural form as parts of the distributed object design. It is a very difficult job for the system designer to assign fixed priorities to the objects and methods, hoping that they will reflect the desired starting time-windows and completion deadlines.

Given natural expressions of timing requirements that the system designer has embedded in the designs of distributed objects, it should be the job of *the middleware, the operating system (OS), and the design tools* to map the designs to the low-level resources, such as processes, threads, I/O devices, and communication network bandwidths, regardless of whether the low-level resources are assigned fixed or dynamic priorities.

Generally, the resource allocation in large-scale systems should involve activities at multiple levels such as the WAN level, LAN level, the computing node level, and the processor level. Therefore, the resource alloca-

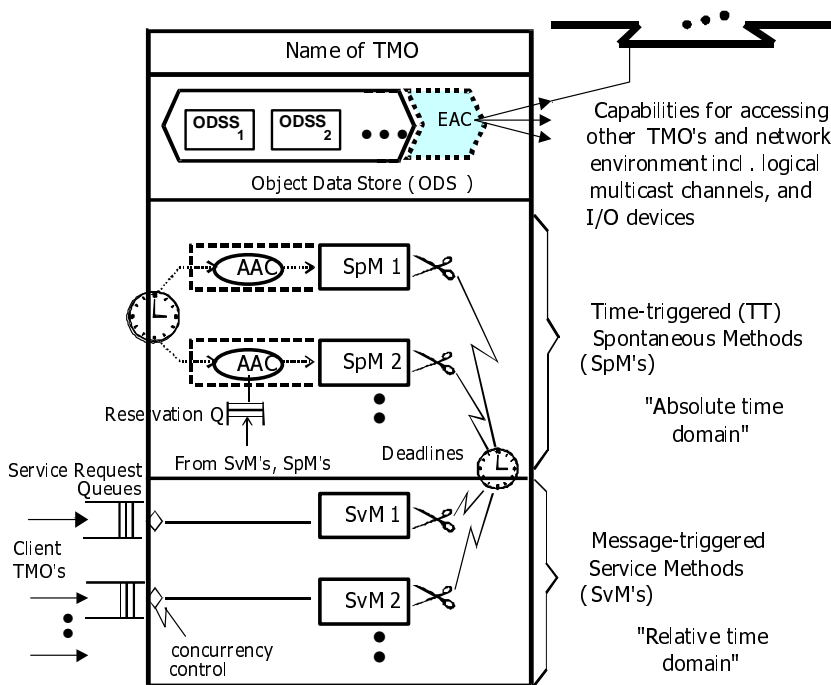


Figure 1. The basic structure of TMO (Adapted from [Kim97a])

tion should be done in a top-down, hierarchical manner, starting from the WAN level and continuing through the LAN and node level down to the processor level.

QoS-driven resource management in RT DC systems has been a research subject for a number of years by now [Kim96, Kim97b, Pal00, Raj97, Raj98]. It is a promising direction but the research is still in its early stage.

In our view, a promising QoS-driven resource allocation approach for the distributed RT systems should start with

- (1) obtaining *rigorous specification of the QoS requirements* associated with application functions, and
- (2) using such specifications *as the main driver for cost-effective resource allocation*.

Ultimately, execution resource requirements come from the needs of producing acceptable-quality outputs of application functions. The most meaningful purpose of any resource allocation is meeting the application requirements with the best quality of execution results and with minimal use of execution resources. Therefore, The effectiveness of a resource allocation must be evaluated on the basis of how well the output quality requirements i.e., QoS requirements of application functions are met after using it.

In most RT applications, component failures must be handled with sufficient efficiency in order not to miss the application objectives. Fault tolerance support

should be an integral part of the resource allocation, because the component failures cannot be totally avoided in a complex system and meeting the fault tolerance requirements involves both static and dynamic allocation of some execution resources.

Application software components are resource users and one of the main purposes of structuring them as RT objects is to structure them in easily analyzable forms. One powerful structuring scheme is the *time-triggered message-triggered object (TMO)* structuring scheme [Kim97a, Kim00a]. It was initially established in early 1990's with a concrete syntactic structure and execution semantics for economical reliable design and implementation of RT systems.

Execution of TMOs can be facilitated by building an execution engine as a middleware running on well-established commercial software / hardware platforms. The TMO scheme enables the distributed RT application designers to express the application's

timing requirements in abstract and natural forms. The current *TMO support middleware (TMOSM)* architecture [Kim99] attempts to honor the application's timing specifications via dynamic resource allocation in a deadline-driven fashion instead of assigning a fixed priority to each object or object method. These are reviewed in Section 2.

A more extensive, QoS requirements specification scheme, named the *risk incursion function (RIF)* scheme, was created by the first co-author in 1994 and since then, slow efforts to convert the concept into a concrete technology have been under way. It can be utilized to improve the efficiency of resource allocation in RT DC systems. This research has been following another complimentary research on a promising middleware architecture, a TMOSM extension called the *RT objected-oriented adaptive fault tolerance support (ROAFTS)* [Kim00b], that supports application software composed of distributed RT objects and also manages execution resources for fault-tolerant system operations. The overall framework of the RIF-based resource management scheme is discussed in Section 3. Three basic practical types of RIFs are then discussed in Section 4 and a design example is also given to show the usage of the RIF scheme in design of RT DC systems.

2. An overview of the TMO scheme and the TMO support middleware (TMOSM)

TMO is a syntactically minor and semantically

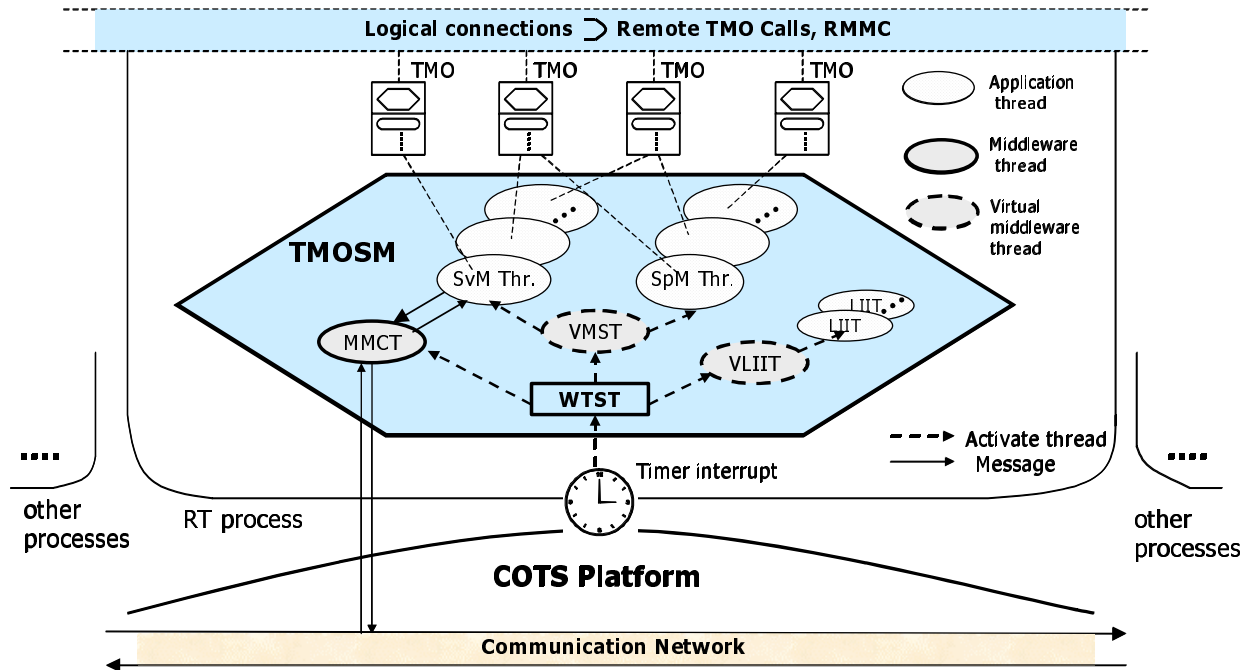


Figure 2. The basic internal thread structure of TMOSM

powerful extension of the conventional object(s). Its structure is depicted in Figure 1. Significant extensions are summarized below and the second and third are the most unique extensions.

- (a) Distributed computing components interacting via remote object method calls;
- (b) Clear separation between two types of methods, *time-triggered methods* (also called *spontaneous methods* or SpMs) and *message-triggered service methods* (SvMs);
- (c) The *basic concurrency constraint* (BCC) that protects SpM executions from indirect disturbance by SvM executions;
- (d) The guaranteed completion time and deadline for result returns.

Details are in [Kim97a, Kim00a].

TMOSM is a TMO execution support middleware architecture which can be easily adapted to most COTS platforms [Kim99, Kim00b]. The internal thread structure of TMOSM is shown in Figure 2. This architecture has been devised to enable relatively easy analysis of the worst-case execution times (WCETs) of TMO methods.

TMOSM consists of three types of threads, *application threads*, *middleware threads* and the *super-micro thread*. TMOSM assigns one application thread to each SpM or SvM of an application TMO. Middleware threads are *periodic threads* (periodically activated to run for a time-slice), each being responsible for a major

part of the functions of TMOSM. The authors believe that structuring of middleware thread as periodic thread is a fundamentally sound approach which leads to easier analysis of the worst-case time behavior of the object execution engine without incurring any significant performance drawback.

The super-micro thread is called the WTST (*Watchdog Timer & Scheduler Thread*). It is a "super-thread" in that it runs at the highest possible priority level. It is also a "micro-thread" in that it manages the scheduling / activation of all other threads in TMOSM. Even those threads created by the node OS before TMOSM starts are executed only if WTST allocates some time-slices to them. Therefore, WTST is activated whenever thread switching needs to be performed, e.g., upon expiration of a time-slice. Also, WTST checks for any deadline violations and if a violation is found, it provides an exception signal to the user.

The three middleware threads function as follows:

- (1) MMCT (Middleware Message Communication Thread): This periodic thread manages the sending of *middleware messages*, which are the messages exchanged among the middleware running on different nodes to support interaction among TMOs, through the communication network. It also distributes middleware messages coming through the network to their destination threads.
- (2) VLIIT (Virtual Local I/O Interface Thread): This virtual thread maintains a pool of threads which are called Local I/O Threads (LIITs) and execute the

I/O requests from application threads. The time-slices allocated to VLIIT are actually distributed to LIITs. Each LIIT is assigned to execute an I/O function that utilizes the I/O capabilities of the host node platform including serial character I/O, disk I/O, network I/O involving messages which are not middleware messages, etc. This VLIIT approach has been motivated by the desire to make it easier to analyze the temporal predictability of application program-segments not involving I/O and the temporal predictability of I/O activities.

- (3) VMST (Virtual Main System Thread): Every time-slice not used by the above two middleware threads is conceptually given to this virtual thread which merely represents all application threads running TMO methods. The actual time-slice allocations are done by WTST that executes the application scheduler function. Therefore, every time-slice conceptually belonging to the VMST is allocated to a fairly selected application thread.

Several features of this TMOSM architecture contribute to simplifying the analysis of the execution time behavior of application TMOs running on TMOSM. First, the strictly periodic nature of middleware threads and the dedication of each middleware thread to a specific functionality enable largely independent analysis of the part of the execution time behavior of application TMOs that depends on a particular middleware thread. For example, in computing the maximum time taken for transmitting a message α in a queue attached to MMCT to a queue attached to MMCT in a remote node, one needs to focus on a few factors only: the number and sizes of the messages in the queue ahead of α , the size of α , guaranteed bandwidth of the network path between the source and the destination, and the frequency and the size of the time-slice given to MMCT.

Secondly, the execution time of an I/O can be in general specified, analyzed, and measured in a larger time-grain than that which can be used in specifying and analyzing the computations relying on the CPU only. Therefore, dedicating LIITs to handling such I/O activities enables the high-precision analysis of the execution time behavior of the CPU-intensive computation.

The first prototype implementation of TMOSM, TMOSM/NT on the Windows NT platform [Ric97], has been developed in the authors' laboratory (DREAM lab of UCI). From the validation tests conducted, we have found that TMOSM/NT can accurately enact the time-window for activating a method as small as 10ms and the method completion deadline as short as 20ms for more than 99.9% of the time [Kim99]. One factor in the TMOSM environment layered on the Windows NT or 2000 platform that is beyond our control and yet can impact the temporal accuracy of TMOSM in a non-negligible manner is the service routines (including

deferred procedure calls) related to interrupts from network interfaces.

The current TMOSM architecture attempts to honor the application's timing requirements via dynamic resource allocation in a deadline-driven fashion. That is, TMOSM schedules the application SpMs and SvMs using the earliest deadline first (EDF) algorithm [Kop97]. If the worst-case execution time (WCET) information for each SpM and SvM is provided by the application designers, the least laxity first (LLF) algorithm can be used instead of EDF.

However, for many complex, safety-critical, distributed RT applications, additional QoS requirements, such as fault tolerance, security, or the relative importance of each output, should be expressed and be considered by the execution engine when the run-time resource allocation decisions are made. In the next section, such a broader QoS requirements specification scheme, named the RIF scheme, is discussed.

Fault tolerance capabilities are required in many distributed RT computing applications. An extension of TMOSM which supports reliable fault-tolerant execution of TMO-structured distributed applications, has been developed in the authors' lab and it has been named the *real-time object-oriented adaptive fault tolerance support* (ROAFTS) [Kim00b, Sho98]. ROAFTS supports fault-tolerant execution of TMOs and conventional passive objects *with tightly bounded time costs*. ROAFTS is aimed for minimizing the efforts of the budget-constrained system engineers to ensure that the risk of the system failing to accomplish its critical missions is minimized.

3. The risk incursion function (RIF) scheme

3.1 RIF: A QoS requirement specification scheme

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 "logical dimension" but also in "time 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 an output to those of the desired output.

System engineers must understand not only the QoS requirements (i.e., output accuracy, fault tolerance), but

also the impacts of QoS losses, i.e., inaccurate outputs, on the overall application success. The potential damaging impacts may be called *risks*. 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 shortages, etc.) and the consequent damage to the application mission is called the *risk incursion function* (RIF) of that output. In short,

RIF := relation (Loss in the timed value accuracy of each output action, Application damage)
 := relation (QoS loss, Risk)

The most common occurrences of the loss in the timed value accuracy are the output actions that are too late. To be general, desired timings of output actions must be expressed in calendar times.

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

- 1) an RIF;
- 2) a specification on the types of fault occurrences that should not be ignored;
- 3) a specification on the types of security threats that should not be ignored;
- 4) a specification of other constraints such as cost constraints, power consumption constraints, and physical volume constraints, etc.

When the accuracy loss Δ_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 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. From this system failure threshold, the system engineer can derive a *serious level* for an individual system output which has the following meaning: If the risk introduced by the inaccuracy of an individual system output exceeds this serious level, it is highly likely that the system failure threshold will also be exceeded during the mission life of the application system. The system failure threshold and the serious level can be used as guidelines for assigning risk incursion values to each system output.

Also, in a certain situation the risk incurred by an inaccurate output may be “erased” after some time called the *validity duration* of the risk. That is, if a burst of inaccurate outputs occurs in a relative short period, the application fails whereas sporadic infrequent occurrences of inaccurate outputs may be automatically compensated by subsequent operations of the application.

3.2 Deriving RIPF from RIF

Each run-time resource allocation procedure is designed to resolve dynamically occurring competitions among various computation-segments for using a certain execution resource. And each run-time resource allocation must be made toward the ultimate goal of minimizing the risks incurred while making minimal uses of execution resources. Therefore, the characteristic description accompanying each resource request presented to a run-time resource allocator must have been derived from the original system-level QoS requirement specifications, i.e., the RIFs for all system outputs.

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 engineers after the task of producing an RIF-based QoS requirement specification. Initially, an RTCS can be represented as a single computation object O which interacts with the external environment by accepting inputs from the sensors and producing outputs to the actuators. Then the design process decomposes the RTCS from a single object into a set of several objects, say $O = \{O_1, O_2, \dots, O_n\}$, which interact with each other as illustrated in Figure 3. Each object produces

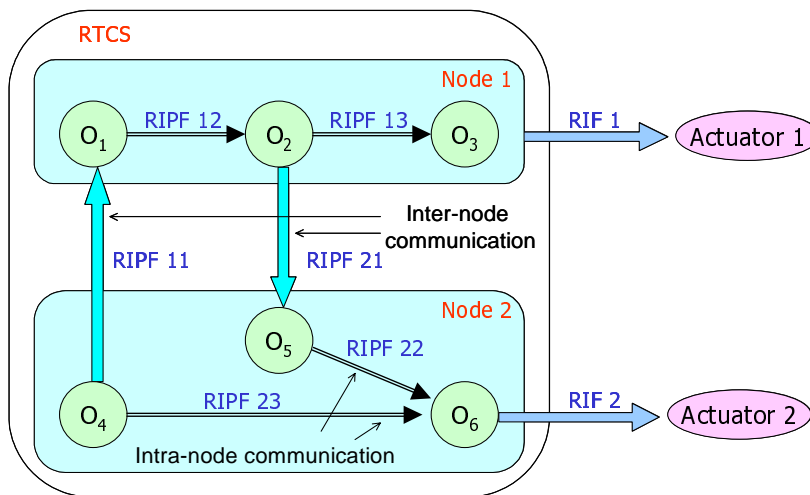


Figure 3. System-level RIF and derived RIPF

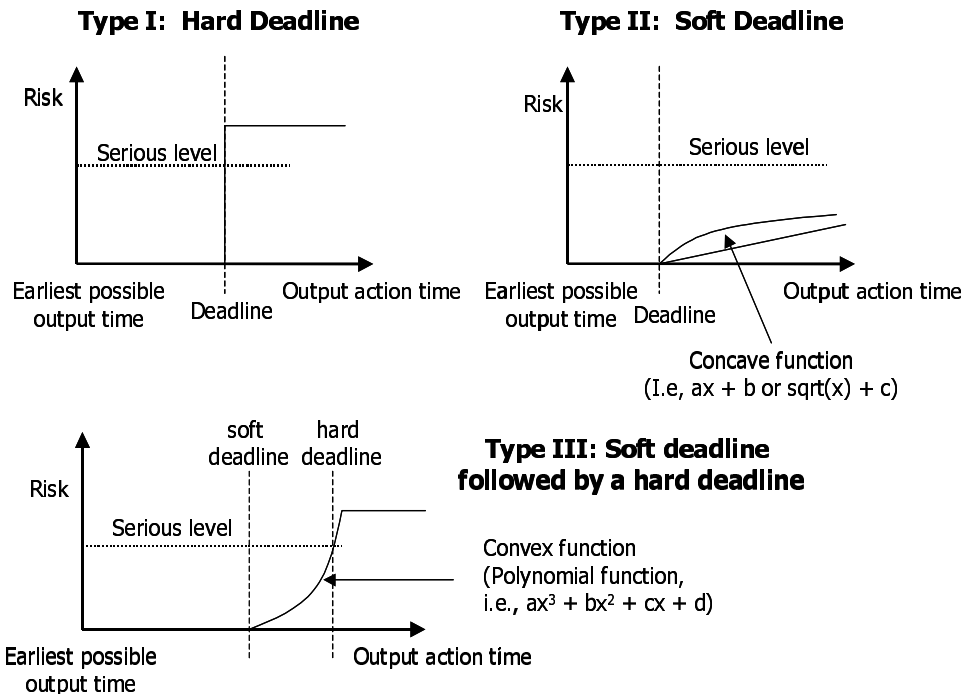


Figure 4. Three basic types of RIF

outputs going to other objects or the environment. Outputs going to other 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* (RIPF), which is a function of the accuracy loss occurred in the intermediate output reaching another object. RIPF is essentially the following relation:

RIPF := relation (Accuracy loss in intermediate output, Potential risk)

Producing an RIPF 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 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:

$\sum_{u_k \in U} \{ \text{risk that can occur through an untimely or inaccurate action of system output } u_k \text{ caused by that untimely execution of } w \}$,

where U is the set of all system output functions. Therefore, an RIPF derived in strict adherence to this

principle will become more complex than the system-level RIFs. The system engineers must produce RIPFs 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 RIPFs in an RIF-based QoS requirement specification is not a mechanical process, various analysis tools that can aid the system engineers in this task are important research subjects.

The decomposition process will continue until the objects become the basic units for resource allocation. In this course of producing multiple levels of objects, RIPFs for the output

functions of each object must be produced. So, RIPFs associated with high-level objects are used as drivers for generating RIPFs associated with low-level objects. In this sense, the RIPFs are produced in a top-down hierarchical manner.

4. Three basic types of RIFs and an example application

In fact, for allocation of CPU time-slices and some other resources, method-segments may be treated as basic users. However, assignment of effective RIPFs for such fine-grain computation units may be very difficult and costly. Therefore, an approach considered to be cost-effective is to associate RIPFs with TMO method completions and each output actions but not with the completions of any smaller computation units. Note that assigning an RIPF to an output action is almost the same as assigning an RIPF to the completion event of the computation-segment that leads to the output.

An RIF-based QoS requirement specification is a solid baseline driver in globally optimal resource allocation. Here we propose three basic RIF types depicted in Figure 4, which we believe are among the most practical and useful types of RIFs. In the diagrams, the y axis shows the risk incurred, and the x axis shows the output action time.

Type I: For output with a hard deadline. The risk immediately jumps from zero and exceeds the serious level after the deadline is violated. The RIF is a step

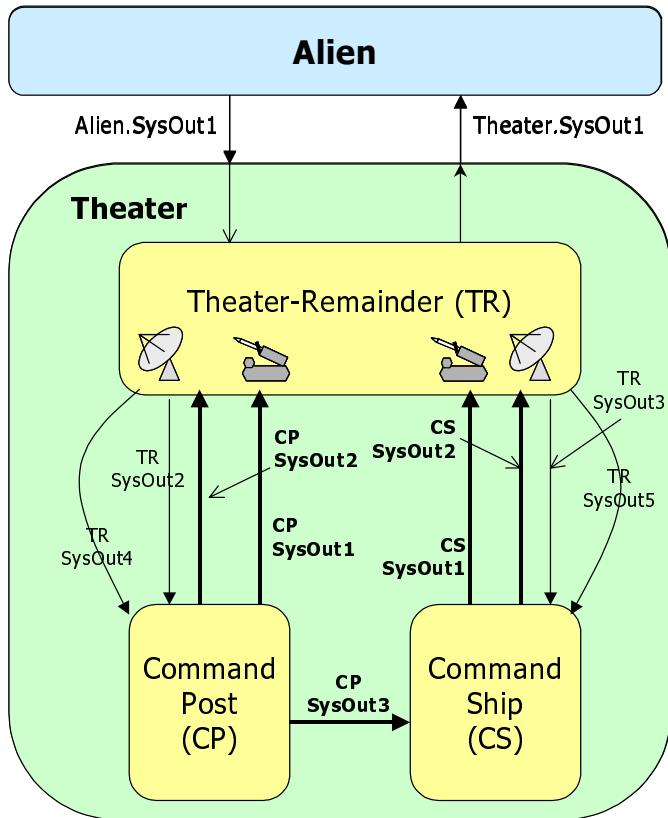


Figure 5. The decomposition of the Theater TMO

function.

Type II: For output with a soft deadline. The risk starts increasing from zero in a concave fashion after the deadline is violated. In this type of RIF, the risk increases so slowly that it may not exceed the serious level within the life time of the application or for the validity duration of the risk.

Type III: For output with a soft deadline followed by a hard deadline. The risk starts increasing from zero in a convex fashion after the soft deadline is violated. Some time after the soft deadline, the risk exceeds the serious level, and that time instant is the hard deadline for this output.

In generating the characteristic descriptions of the resource requests of various computation-segments, i.e., RIFs, 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 engineers. One such representation approach is the TMO structuring scheme. 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 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. Here we will briefly present an example of the RIF-based system design procedure in the context of TMO-structured top-down design.

CAMIN (Coordinated Anti-Missile Interceptor Network) is a model of a defense command-and-control network which has been used in the authors' laboratory as an example of an advanced RT distributed computing application. For more detailed description of the functionality of CAMIN, readers are referred to [Kim97a]. The application environment in CAMIN is a sky, land, and sea segment (together called *Theater*) – in which moving items are taken seriously. The moving items include at least one valuable target to be defended (that is assumed to be the command ship in the sea here) and flying items, both hostile and non-threatening. The high-level requirements are:

- 1) Each flying object should be intercepted if it is considered dangerous; and
- 2) The valuable target (the command ship) can move around to avoid the dangers posed by flying items.

In the first step, the system engineer describes the application as two TMO's, the Theater TMO and the Alien TMO. Alien has one system output, Alien.SysOut1, which sends missiles and non-threatening flying objects (NTFOs) to Theater. Theater also has one system output going to Alien, Theater.SysOut1, which sends information about the defense targets and the current status of the missiles and commercial airplanes leaving from Theater to the outside. In a sense, Alien determines the workload for Theater.

In the next step, the Theater TMO can be further delineated and decomposed into three TMO's, Army's Command Post TMO (CP), Command Ship TMO (CS), and Theater-Remainder TMO (TR), as shown in Figure 5. The system outputs are:

- CP.SysOut1: Send an intercept order to TR.
- CP.SysOut2: Send a radar spot-check plan to TR.
- CP.SysOut3: Hand-over of data on dangerous items to CS.
- CS.SysOut1: Send an intercept order to TR.
- CS.SysOut2: Send a radar spot-check plan to TR.
- TR.SysOut2: Send radar data (both from a scan and a spot-check) to CP.
- TR.SysOut3: Send radar data (both from a scan and a spot-check) to CS.

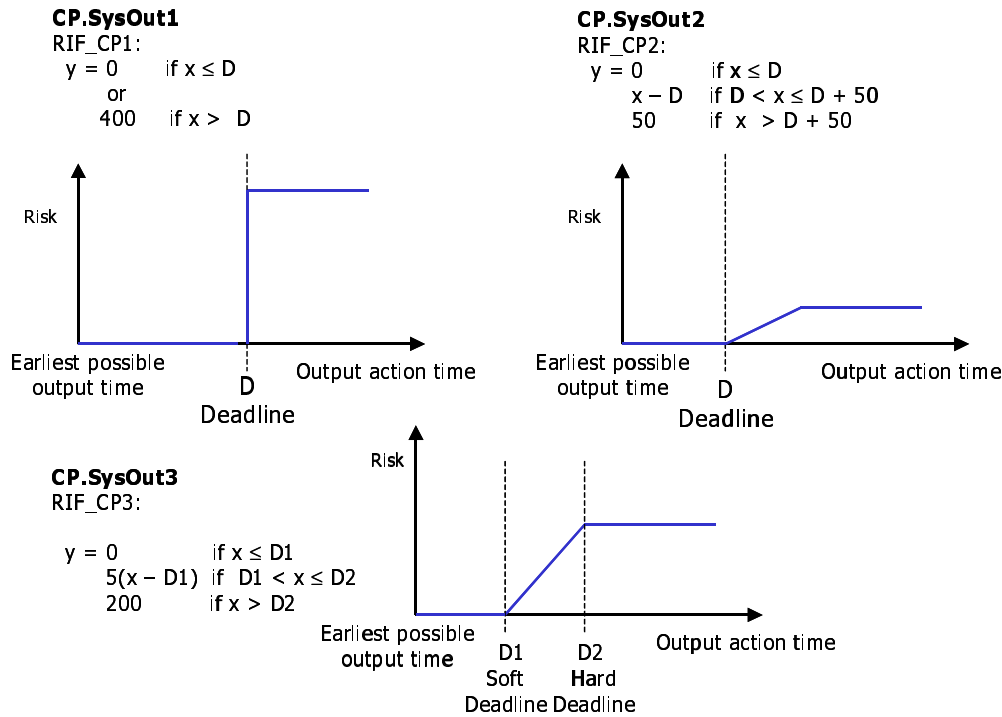


Figure 6. The RIFs for the Command Post (CP) TMO

Assume that when Alien throws enemy missiles into Theater, it always puts them in the territory covered by the Army's CP and lets them move toward CS. After receiving the radar data from TR, CP must send CP.SysOut2 in time to require more detailed radar data, the spot-check data, to track the enemy missiles. After some tracking, CP must send CP.SysOut1 in time, which is the interception order, because otherwise the interception will be a failure. Since the radar in the command ship (CS TMO) is a short-range radar, CP also needs to send some warning data about dangerous surviving enemy missiles to the CS in advance. Otherwise, CS might not have enough time to intercept the missiles once CP misses them. Therefore, CP uses CP.SysOut3 to send the warnings with historical data on dangerous items to CS. Given these functionalities, a hard deadline may be imposed on CP.SysOut1, and a soft deadline on CP.SysOut2, while CP.SysOut3 may have a soft deadline followed by a hard deadline. Also CP.SysOut1 has higher risk value than CP.SysOut2 and CP.SysOut3 do.

The deadlines for these system outputs are decided based on the consideration of physical constraints. For example, the physical constraints include the frequency and accuracy of the radar data sent to CP, the flying speeds of the interceptors and the missiles, etc.

Suppose the system failure threshold is set to be 1000. Derived from this, the risk value of CP.SysOut1 after its hard deadline is missed is set to 400, the

maximum risk value of CP.SysOut2 is set to 50, and the risk value of CP.SysOut3 after its hard deadline is missed is set to 200. Figure 6 shows the RIFs for CP's system outputs.

After the RIFs are decided, the next step is to further delineate and decompose the CP to multiple TMO's and derive an RIPF for each new TMO. The derivation of RIPFs from an RIF is based on the worst-case execution time analysis and the way the output of each TMO is used by other TMOs or actuators. Determining RIPFs for TMOs is practically to determine RIPFs for SpMs and SvMs, including both method completion actions and

specific output actions of SpMs and SvMs. After reaching this step, the procedure for deriving RIPFs is completed. The derived RIPF set can be used by resource allocators, such as the processor scheduler, the communication network bandwidth scheduler, and the I/O devices scheduler.

Since the derived RIPF set also incorporates deadline information for each SpM and SvM, the RIPF-driven resource schedulers can schedule various resources at least as efficiently as the deadline-driven resource schedulers do. One example algorithm of the RIPF-driven processor scheduler may function as follows:

- 1) Run the LLF algorithm, and if a zero-risk arrangement can be found, i.e., the deadlines of all tasks can be met, use this arrangement;
- 2) Otherwise, schedule tasks with higher risk values first, which means the tasks with low risks might be sacrificed.

This RIPF-driven algorithm cannot perform worse than LLF does. In the case where not all deadlines can be met under LLF, it might do a better job by considering the potential risk values together with the deadline information in the RIPFs.

5. Conclusion

The TMO scheme enables the distributed RT

application designers to express the application's timing requirements explicitly in a natural form. A scheme for specification of not just timing and other QoS requirements associated with system output actions, but also the potential damages that failing to meet the requirements bring in, has been evolving in the authors' laboratory under the name of the risk incursion function (RIF) scheme. The essence of the current version of the scheme was discussed in this paper, and a recently refined approach for derivation of RIFs from the original RIF-based QoS requirements specification was also discussed. The TMOSM architecture described is being expanded to incorporate the RIF-driven schedulers. In addition, three basic types of RIFs which we believe to be among the most practically useful RIFs were presented.

The RIF-based resource allocation will result in minimizing the application risks, or to put it differently, will lead to maximizing the survival periods of the distributed RT applications. The research on QoS-driven resource allocation in complex distributed RT systems is still in its early stage. In order to establish it as a widely practicable technology, much further research is needed on systematic derivation of RIFs from the original RIF-based QoS requirements specification. RIF-driven resource allocation algorithms as well as their efficient incorporations into the middleware architecture are also subjects for extensive research in the future.

Acknowledgements: The research on the DTS scheme was supported earlier by the US NSWC and the US DARPA under Contract N66001-97-C-8516. The current work is supported in part by the NSF under Grant Numbers 99-75053 (NGS) and 00-86147 (ITR), and in part by the US DARPA under Contract F33615-01-C-1902 monitored by AFRL. No part of this paper represents the views and opinions of any of the sponsors mentioned above.

References

[Bol00] Greg Bollella and James Gosling, "The Real-Time Specification for Java", *IEEE Computer*, June, 2000, pp. 47-54.

[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., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No.8, August 1997, pp. 62-70.

[Kim97b] Kim, K.H., "Toward Globally Optimal Resource Management in Large-Scale Real-Time Distributed Computer Systems", *Proc. FTDCS '97*

(*IEEE CS 6th Workshop on Future Trends of Distributed Computing Systems*), Tunis, Tunisia, Oct. 1997, pp.248-255.

[Kim99] Kim, K.H. Ishida, Masaki, Liu, Juqiang, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. 2nd IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC '99)*, St. Malo, France, May, 1999, pp.54-63.

[Kim00a] Kim, K.H., "APIs for Real-Time Distributed Object Programming", *IEEE Computer*, June 2000, pp.72-80.

[Kim00b] Kim, K.H., "Object-Oriented Real-Time Distributed Programming and Support Middleware", *Proc. ICPADS 2000 (7th Int'l Conf. on Parallel & Distributed Systems)*, Iwate, Japan, Jul. 2000, pub. by IEEE CS Press (keynote paper), pp.10-20.

[Kop97] Kopetz, H., 'Real-Time Systems: Design Principles for Distributed Embedded Applications', *Kluwer Academic Publishers*, ISBN: 0-7923-9894-7, Boston, 1997.

[OMG01] Object Management Group, "The common Object Request Broker: Architecture and Specification", Revision 2.5, Sep., 2001.

[Pal00] Pal, P.P., Loyall, J.P., Schantz, R.E., Zinky, J.A., Shapiro, R, Megquier, J., "Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration", *Proc. ISORC 2000 (3rd IEEE Int'l Symp. on Object-Oriented Real-time Distributed Computing)*, March, 2000, Newport Beach, CA, pp. 310-319.

[Raj97] Raj Rajkumar, Chen Lee, John Lehoczky and Dan Siewiorek, "A Resource Allocation Model for QoS Management", *Proc. 1997 IEEE Real-Time Systems Symp.*, San Francisco, CA, December 1997.

[Raj98] Raj Rajkumar, Chen Lee, John Lehoczky and Dan Siewiorek, "Practical Solutions for QoS-based Resource Allocation Problems", *Proc. 1998 IEEE Real-Time Systems Symp.*, Madrid, Spain, 1998.

[Ric97] Richter, J., 'Advanced Windows', 3rd Edition, *Microsoft Press*, 1997.

[Sho98] Shokri E, Crane, P., and Kim, K.H., and Subbaraman, C., "Architecture of ROAFTS/Solaris: A Solaris-based Middleware for Real-Time Object-Oriented Adaptive Fault Tolerance Support", *Proc. COMPSAC '98 (IEEE CS 22nd Int'l Computer Software & Applications Conf.)*, Vienna, Austria, August 1998, pp.90-98.