Real-Time Communication using TMO(Time- Triggered and Message-Triggered Object) in Distributed Computing Systems

Gwang-Jun Kim, Chun-Suk Kim, Yong-Gin Kim, Chan-Ho Yoon and Moon-Hwan Kim *Member, KIMICS*

Abstract—Real-time(RT) object-oriented(OO) distributed computing is a form of RT distributed computing realized with a distributed computer system structured in the form of an object network. Several approached proposed in recent years for extending the conventional object structuring scheme to suit RT applications, are briefly reviewed. Then the approach named the TMO (Time-triggered Message-triggered Object) structuring scheme was formulated with the goal of instigating a quantum productivity jump in the design of distributed time triggered simulation. The TMO scheme is intended to facilitate the pursuit of a new paradigm in designing distributed time triggered simulation which is to realize real-time computing with a common and general design style that does not alienate the main-stream computing industry and yet to allow system engineers to confidently produce certifiable distributed time triggered simulation for safety-critical applications. The TMO structuring scheme is a syntactically simple but semantically powerful extension of the conventional object structuring approached and as such, its support tools can be based on various well-established OO programming languages such as C++ and on ubiquitous commercial RT operating system kernels. The Scheme enables a great reduction of the designers efforts in guaranteeing timely service capabilities of application systems

Index Terms—Guaranteeing time, Object-Oriented distributed Computing, Real-time operating kernel, Time-triggered Message- triggered Object (TMO)

I. INTRODUCTION

ONE of the computer application fields which started showing noticeable new growth trends in recent years is the real-time(RT) computing application field. RT simulator developments are under increasing demands[1]-[4]. For example, continuing advances in virtual reality application accompany increasing demands for more powerful RT simulation capabilities. Numerous other examples can also be found in the RT computing control field. Not only description but also simulation of application environments is often performed as integral steps of validating control computer system designs. RT simulators of application environments can often enable highly cost-effective testing of the control computer systems implemented. Such testing can be a lot cheaper than the testing performed in actual application environments while being much more effective than the testing based on non-RT simulators of environments.

Future RT computing must be realized in the form of a generalization of the non-RT computing, rater than in a form looking like an esoteric specialization. In other words, under a properly established RT system engineering methodology, every practically useful non-RT computer system must be realizable by simply filling the time constraint specification part with unconstrained default values.

The current reality in RT computing is far from this desirable state and this is evidenced whether one looks at the subfield of operating systems or that of software/ system engineering tolls. Another issue of growing importance is to provide in the future an order-of-magnitude higher degree of assurance on the reliability of distributed time triggered simulation products than what is provided today. To require the system engineer to produce design-time guarantees for timely service capabilities of various subsystems which take the form of objects in OO system designs.

The major factor that has discouraged any attempt to do this has been the use of software structuring approaches and program execution mechanisms and modes which were devised to maximize hardware utilization but at the cost of increasing the difficulty of analyzing the temporal behavior of the RT computation performed.

Most concerns were given to the issue of how to maximally utilize uniprocessor hardware even at the cost of losing service quality predictability. System engineers were more willing to ignores a small percentage of peak-load situations which can occur and can lead to excessively delayed response of distributed time triggered simulation, instead of using more hardware-consuming design approaches for producing timeliness-guaranteed systems.

Manuscript received January 3, 2003

G. J. Kim is with the Computer Engineering Department, University of Yosu National, Chonnam, 550-749 Korea, (corresponding author to provide phone: +82-061-659-3255; fax: +82-061-659-3250; e-mail: Kgj@ yosu.ac.kr).

C. S. Kim is with the Electronic Communication Engineering Department, University of Yosu National, Chonnam, 550-749 Korea, (e-mail: csKim@ yosu.ac.kr).

Y. G. Kim is with the Electrical Engineering Department, University of Jeonnam National, Gwangju, 500-757 Korea, (e-mail: yjk2910 @lgcaltex. co.kr).

C. H. Yoon is with the Computer Engineering Department, University of Chosun, Gwangju, 501-759 Korea, (e-mail: Yoonchanho@hotmail.com).

M. H. Kim is with the Computer Engineering Department, University of Chosun, Gwangju, 501-759 Korea, (e-mail: kmh@ktf.com).

II. DISTRIBUTED REAL TIME COMMUNICATION FRAMEWORK FOR SYSTEMATIC DEADLINE HANDLING

Fig. 1 depicts the relationship between a client and a server component in a system composed of hard real time components which are structured as distributed computing objects. The client object in the middle of executing its method, Method1, calls for a service, Method 7 service, from the server object. In order to complete its execution of Method 1 within a certain target amount of time, the client must obtain the service result from the server within a certain deadline.

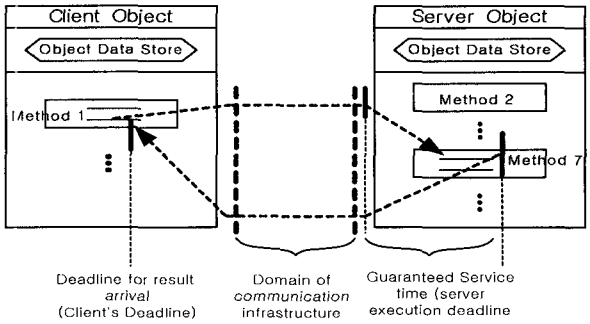

Fig. 1 Client's deadline vs Server's guaranteed service time

This client's deadline is thus set without consideration of the speed of the server. During the design of the client object, the designer searches for a server object with a guaranteed service time acceptable to it.

Actually the designer must also consider the time to be consumed by the communication infrastructure in judging the acceptability of the guaranteed service time of a candidate server object.

In general, the following relationship must be maintained:

Time consumed by communication infrastructure + Guaranteed service time
< Maximum transmission times imposed on communication infrastructure + Guaranteed service time < Deadline for result arrival- Call initiation instant

where both the deadline imposed by the client for result arrival and the initiation instant of the client's remote service call are expressed in terms of absolute real time, e.g., 10am.

There are three sources from which a fault may arise to cause a client's deadline to be violated. They are (s1) the client object's resources which are basically node facility, (s2) the communication infrastructure, and (s3) the server object's resources which include not only node facility but also the object code. The server is responsible to finish a service within the guaranteed service time, while the client is responsible for checking if the result comes back within the client's deadline. Therefore, the client object is responsible for checking the result of the actions by all the resource involved, whereas the server object is responsible for checking the result of the actions of (s3) only.

Since an RT simulator must exhibit the timing behavior which is the same as or very close to that of the simulation target, the simulator clock must tick at a steady rate. Each tick of the simulator clock is commenced and administered by referencing an RT clock in the simulation execution engine. The ticking rate of the simulator clock in an RT simulator must be chosen such that during any ticking interval, all events which should be simulated during that interval may be transparent to the user of the RT simulator and only the resulting state of the simulator at the end of the ticking interval may be seen by the user. Therefor, the microscopic order in which events are simulated during a ticking interval should be of no concern to the simulator user. This requirement is called the simulator clock atomicity requirement[5]-[6]. All computational activities taking place during a ticking interval of the simulator clock maybe viewed as one simulation-step.

In distributed RT simulation, simulator objects are distributed among multiple nodes. Synchronization of the simulation steps of distributed simulator objects in then a key challenge. In other words, a simulation step executed by every member of the distributed simulator object group must be synchronized with the corresponding simulation step executed by any other member. In other word, the simulator clock for one simulator object must commence the n-th tick neither before the (n-1)th tick by the clock for another simulator object nor after the (n+1)-th tick by the clock for another simulator object.

Therefore, every member must perform some activities necessary to stay synchronized with other members. For example, distributed nodes may exchange completion reports at the end of each simulation-step. However, this is not an efficient approach when the number of nodes used is large. The essence of the distributed time-triggered simulation approach is the following:

(1) Every node is equipped with an RT clock and executes each simulation step upon reaching of the RT clock at the predetermined value; and
(2) Every simulation step is designed to be completed within on ticking interval.

The distributed time triggered simulation approach has major advantages over other distributed simulation approaches, even if we assume that the latter approaches can be adapted somehow to enable RT simulation. This is because synchronization of simulation steps executed by distributed simulator objects under the distributed time triggered simulation scheme does not require message exchanges among the host nodes. The advantages become decisive in heavy load distributed simulation situations.

However, even with the distributed time triggered simulation approach, exchanges of message that represent movements of certain simulation targets form the territory covered by one TMO to the territory covered by another TMO are inevitable. Therefore, the ticking interval must be long enough to cover this kind of message exchanges.

III. AN OVERVIEW OF THE TMO SCHEME

As a concrete example of a high-level OO RT distributed programming approach that has been based on the philosophy discussed in the preceding section, the time-triggered message-triggered object (TMO) programming scheme is briefly summarized in this section[2]-[6].

The TMO scheme was established in early 1990's with a concrete syntactic structure and execution semantics for economical reliable design and implementation of RT systems. The TMO scheme is a general-style component structuring scheme and supports design of all types of components including distributable objects and distributable non-RT objects within one general structure.

Calling the TMO scheme a high-level distributed programming scheme is justified by the following characteristics of the scheme:

(1) No manipulation of processes and threads : Concurrency is specified in an abstract form at the level of object methods. Since processes and threads are transparent to TMO programmers, the priorities assigned to them, if any, are not visible, either.

(2) No manipulation of hardware-dependent features in programming interactions among objects : TMO programmers are not burdened with any direct use of low-level network protocols and any direct manipulation of physical channels and physical node addresses/ names.

(3) No specification of timing requirements in (indirect) terms other than start-windows and completion deadlines for program units (e.g., object methods) and time-windows for output actions : TMOs are devised to contain only high-level intuitive and yet precise expressions of timing requirements. Priorities are attributes often attached by the OS to low-level program abstractions such as threads and they are not natural expressions of timing requirements. Therefore, no such indirect and inaccurate styles of expressing timing requirements are associated with objects and methods.

At the same time the TMO scheme is aimed for enabling a great reduction of the designer's efforts in guaranteeing timely service capabilities of distributed computing application systems. It has been formulated from the beginning with the objective of enabling design-time guaranteeing of timely actions. The TMO incorporates several rules for execution of its components that make the analysis of the worst-case time behavior of TMOs to be systematic and relatively easy while not reducing the programming power in any way.

TMO is a natural and syntactically minor but semantically powerful extension of the conventional object(s)[6].

As depicted in Fig. 2 the basic TMO structure consists of four parts:

ODS-sec = object-data-store section : list of object-data-store segments(ODSS's);

EAC-sec = environment access-capability section : list of gate objects (to be discussed later) providing efficient call-paths to remote object methods, logical communication channels, and I/O device interfaces;

SpM-sec = spontaneous-method section : list of spontaneous methods;

SvM-sec = service-method section.

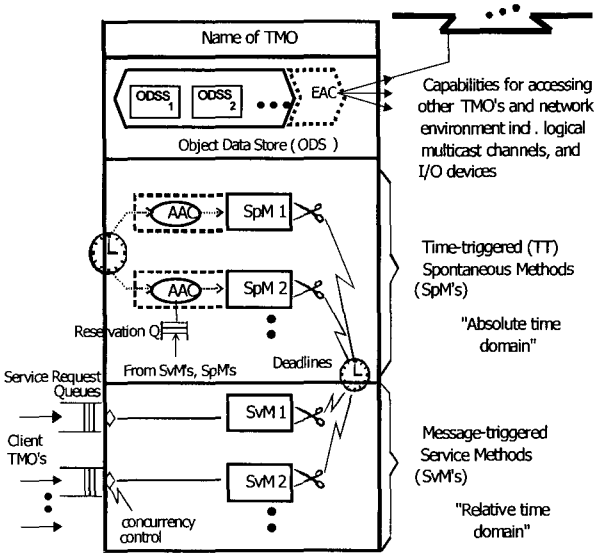

Fig. 2 Structure of the TMO

Major features are summarized below.

(a) Distributed computing component :

The TMO is a distributed computing component and thus TMOs distributed over multiple nodes may interact via remote method calls. To maximize the concurrency in execution of client methods in one node and server methods in the same node of different nodes, client methods are allowed to make non-blocking types of service requests to server methods.

(b) Clear separation between two types of methods:

The TMO may contain two types of methods, time-triggered (TT-) methods (also called the spontaneous methods of SpMs), which are clearly separated from the conventional service methods (SvMs). The SpM executions are triggered upon reaching of the RT clock at specific values determined at the design time whereas the SvM executions are triggered by service request messages from clients. Moreover, actions to be taken at real times which can be determined at the design time can appear only in SpMs.

(c) Basic concurrency constraint (BCC):

This rule prevents potential conflicts between SpMs and SvMs and reduces the designer's efforts in guaranteeing timely service capabilities of TMOs. 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 when and execution time-window big enough for the SvM that does not overlap with the execution time-

window of any SpM that accesses the same ODSSs to be accessed by the SvM, opens up. However, the BCC does not stand in the way of either concurrent SpM executions of concurrent SvM executions.

(d) Guaranteed completion time and deadline:

The TMO incorporates deadlines in the most general form. Basically, for output actions and method completions of a TMO, the designer guarantees and advertises execution time-windows bounded by start time and completion times.

Triggering times for SpMs must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section. An example of an AAC is

"for t = from 10am to 10:50am every 30min
start-during (t, t+5min) finish-by t+10min"

which has the same effect as

"start-during (10am, 10:05am)
finish-by 10:10am",
"start-during (10:30am, 10:35am)
finish-by 10:40am"

A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM within the same TMO object requests future executions of a specific SpM. Each AAC specifying candidate triggering times rather than actual triggering times has a name.

An underlying design philosophy of the TMO scheme is that an RT computer system will always take the form of a network of TMOs. The designer of each TMO provides a guarantee of timely service capabilities of the object. The designer does so by indicating the guaranteed execution time-window for exery output produced by each SvM as well as by each SpM executed on requests from the SvM and the guaranteed completion time (GCT) for the SvM in the specification of the SvM. Such specification of each SvM is advertised to the designers of potential client objects. Before determining the time-window specification, the server object designer must convince himself/herself that with the object execution engine (a composition of hardware, node OS, and middleware) available, the server object can be implemented to always execute the SvM such that the output action is performed within the time-window. The BCC contributes to major reduction of these burdens imposed on the designer.

Middleware which together with node OSs and hardware make up TMO execution engines, have been developed.

IV. RT OBJECT STRUCTURING TOOL AND THE TMO APPROACHE

In this section, major desirable capabilities of a full-featured RT object structuring tool are discussed along with the approaches adopted in the TMO scheme to realize such capabilities. This discussion, together with the overview given in Sec. 3, reveals almost all the important features of the TMO scheme.

Clear specification of timing constraints is a fundamental requirement in rigorous engineering of RT computer systems. Major issues in this area are:

(1) Global time base;
(2) Time-triggered (TT) action; and
(3) Separation of the absolute time domain from the relative time domain

A. Global time base

In any practical RT system design or programming language, the following features must be included:

(1) Specification of time bases: This includes specifying UTC (Universal Time Coordinated), SST (the time elapsed since the distributed system started), etc.
(2) Global-time reference function: This includes now which returns the current time obtained from the global time base, forever which is a time constant representing a practically infinite time interval, etc.

Naturally, the TMO scheme provides these facilities.

B. Time-triggered(TT) action

Specification of TT computations is a fundamental feature of RT programming that distinguishes RT programming from non-RT programming. The computation unit can be any one of the following:

(1) Simple statement such as an assignment statement with the right-side expression restricted to an arithmetic logical expression type involving neither a control flow expression nor a function call, an I/O command statement, etc.;
(2) Compound statement such as if-then-else statement, while-do statement, case statement, etc.;
(3) (Statement) Block;
(4) Function and Procedure;
(5) Object method

TT actions associated with a computation unit may include TT initiation of the computation unit, timely completion of the computation unit, and periodic execution. Therefore, in any practical RT system design or programming language, it is desirable to have the following type of a construct:

```
ab      "timing specification begin"
        for <time-var> = from <activation-time>
              to <deactivation-time>
              [every <period>]
        start-during
              (<earliest-start-time>,
              <latest-start-time>)
```

finish-by <deadline>
ae "timing specification end"

For example, consider the following case.

"for t = from 10am to 10:50am every 30 min start-during (t, t+5min) finish-by t+10 min"

This specifies: "The associated computation unit must be executed every 30 minutes starting at 10am until 10:50am and each execution must start at any time within the 5minute interval (t, t+5min) and must be completed by t+10min."

So, it has the same effect as

{"start-during (10am, 10:05am) finish-by 10:10am", "start-during (10:30am, 10:35am) finish-by 10:40am"}.

of the five types of computation nits mentioned above, the object method is the most frequently used unit for TT initiations and completion time checks. The TMO execution engines built so far fully allow the specifications of TT initiations, completion deadlines, and periodic executions to be associated with object methods but only to limited extent TT executions of segments of object methods. In other words, object methods, SpM's and SvM's, are about the only basic schedulable computation units fully supported so far. This does not seriously limit the programming power and flexibility offered to RT programmers and yet greatly simplifies the job of constructing reliable efficient execution engines. However, there is no intrinsic limitation of the TMO structure that prevents the incorporation of TT initiation into other computation units. Such an extension just requires construction of TMO execution engines capable of accurately scheduling finer-grain RT computation units.

To support TT executions of method segments in a limited form, an SpM may contain

"at global-time-constant do S" and
"after global-time-constant do S"

statements, where global-time-constant must be an RT instance preceding the completion deadline of the SpM. Such statements can be executed by the execution engine without incorporating any new major OS (scheduler) parameters. A simple OS service such as "yield the current time-slice of mine to another thread if global-time constant is more than one time-slice away from now", can be easily implemented and support the statements well.

C. Separation of the absolute time domain from the relative time domain

From the viewpoint of obtaining easily understandable and analyzable RT programs, it is also good to clearly separate the specification of the computation dealing with the absolute time domain, i.e., the computation dependent of the time-of-day information available from the global time base, from the specification of the computation dealing with the relative time domain only. In the case of the TMO structuring scheme, SvM's deal with the relative time domain only, i.e., they use only the elapsed intervals since the method was started by an invocation message from an object client. This is natural since the arrival time of a service request (i.e., a message invoking a service method) from an object client cannot be predicted by the designer of the service method in general, especially when that designer is not the designer of the client object. Therefore, with one exception to be discussed below, any use of the time-of-day information can be used within SpM's only. This means that computations of the type

"at global-time-constant do S" or
"after global-time-constant do S"

can appear only in SpM's.

The only exception allowed is that an arithmetic logical expression consisting of now and global time constants may be used in a server method for the purpose of selecting candidate triggering times associated with SpM's

V. INTERACTION AMONG OBJECTS AND MESSAGE FOR RT COMMUNICATION

A. Non-blocking call

An underlying designs philosophy of the RT OO distributed computing approaches is that every RT DCS will be designed in the form of a network of RT objects. RT objects interact via calls by client objects for service methods in server objects. The caller may be a TT method or a service method in the clients. In order to facilitate highly concurrent operations of client and server objects, non-blocking (sometimes called asynchronous) types of calls(i.e., service request) in addition to the conventional blocking type of calls service methods should be allowed. Therefore, the TMO scheme supports the following two basic types of calls to service methods in the server TMO.

(1) Blocking call: After calling a service method, the client waits until a result message in returned from the service methods. The syntactic structure may be in the form of

Obj-name. SvM-name(parameter-1, parameter-2,..., by deadline).

Since the client and the server object may be resident in two different processing nodes, this call is in general implemented in the form of a remote procedure call. Even if there is no result parameter in the service method, the execution completion signal from the server method does not arrive by the specified deadline, then the execution engine for the client object invokes an appropriate exception handling function as it would when an arithmetic overflow occurs.

(2) Non-blocking call: After calling a service method, the client can proceed to follow-on steps(i.e., statements or instructions) and then wait for a result message from the service method. The syntactic structure may be in the form of

Obj-name.SvM-name(parameter-1,parameter-2,..., mode NWFR, timestamp TS);
-------statements------;
get-result Obj-name.SvM-name(TS) by deadline;

The mode specification "NWFR" which is an abbreviation of "No-Wait-For-Return" indicates that this is a non-blocking call. When the client calls the service method, the clients records a time-stamp into a variable, say TS. The time-stamp uniquely identifies this particular call for the service method from this client. Therefore, later when the client needs to ensure by execution of the "get-result" statement the arrival of the results returned form the earlier non-blocking call for the service method, not only the service method name but also the variable TS containing the time-stamp associated with the subject call must be indicated. When a client make multiple non-blocking calls for service methods before executing a "get-result" statement, the time-stamp unambiguously indicate to the execution engine which non-blocking call is referred to. If the results have not been returned at the time of executing the "get-result" statement, the client waits until the execution engine recognizes the arrival of the results. A non-blocking call thus creates concurrency between a client method (TT method or service method) and a service method in a server object and the concurrency lasts until the execution of the corresponding "get-result" statement. In some situations, a client does not need any result form a non-blocking call for a service method. Such a client does not use a "get-result" statement.

B. Client-transfer call

Even though its needs were initially recognized in the context of the TMO scheme, it is of fundamental nature and may be useful in almost all types of RT systems, Basically, an SvM in a TMO may pass a client request to another SvM by using a client-transfer call. The latter SvM may again pass the client request to another SvM. This chaining sequence may repeat until the last SvM in the chain returns the results to the client. The main motivation behind such a client-transfer call stamp from BCC which require an execution of an SVM to be made only if a sufficiently large time-window between executions of SpM's potentially conflicting with the SvM opens up. Hence, in certain situations a highly complicated SvM may never be executed due to the lack of a wide enough time-window. One way to get around this problem is to divide the SvM into multiple smaller SvM's, SvM1,..., SvMx. A client can then call each smaller SvM each time. Calling each smaller SvM incurs the communication overhead of transmitting a request to the smaller SvM and obtaining the results. Substantial reduction of such communication overhead is the motivation behind an arrangement in which the client calls the first SvM and the latter passes the "service contract" with the client on to another SvM and so on until the last SvM of the chain returns the results to the client.

As a part of executing this client-transfer call for an SvM, the execution engine terminates the caller SvM, places a request for execution of the called SvM into the service request queue for the called SvM, and establishes the return connection from the called SvM to the client of the caller SvM that has just been terminated. When the "return" statement in the called SvM is executed, the results are returned through the return connection established. Since the external clients which called the first SvM cannot predict form which SvM it will receive returned results, it is implemented to accept results without having to know which SvM the results came form.

A client-transfer call may involve passing parameters in an explicit manner as done in the case of a call by an external client or passing information through the shared data structures in the ODS. The syntactic structure for such a client-transfer call for an SvM may be in the form of

ClientTransferCall(SvM_name, parameters)

SvM_name identifies the SvM being called. The ID of the port or channel through which the current client of the caller SvM is prepared to receive return results is passed by the object execution engine onto the execution support record for the SvM being called. That is, a proper return connection is established between the SvM being called and the current client of the caller SvM. The parameters may include the parameters newly created by the caller SvM as well as those created by predecessors in the client-transfer chain.

There is no reason why this client-transfer call cannot be extended to the case of calling an SvM in another TMO. The syntactic structure for such a client-transfer call for an external SvM is about the same, i.e.,

ClientTransferCall(Obj_name.SvM_name,parameters)

In fact, there is no essential need for a client to distinguish between the case where results are returned form the called SvM and the case where results are returned form another SvM.

Actually, one can take the view that accepting results returned form the called SvM is a special case of accepting results returned form any SvM in the system.

C. RT message communication and programmable multicast channels

Whether a service request is a blocking call or a non-blocking call, the request message and the result return message must be communicated with predictable delay bounds. May protocols suitable for RT message communication over local area networks and wide area networks exist, e.g., time-division multiplexed access (TDMA), token ring access, deterministic CSMA/CD, ATM, etc.

In addition to the interaction mode based on remote method invocations, distributed RT objects can use another interaction mode where messages may be exchanged over message channels explicitly specified as

data members if involved objects. See Fig. 2 For example, logical multicast channels, LMC1 and LMC2, can be declared as data members of each of the three remotely cooperating RT objects, TMO1, TMO2, and TMO3, during the design time. The compiler and the object execution engines running the tree RT objects must then together facilitate the two channels and guarantee timely transmission of message over those channels. Once TMO1 sends a message over LMC1, ten the message will be delivered to the ODS of each of the three RT objects. Later during their execution certain methods in TMO2 and TMO3 can pick up the messages that came over LMC1 into the ODS's of their host objects. In many applications, this interaction mode leads to better efficiency than the interaction mode based on remote method invocations does.

D. Deadline specification and service time guarantee

As mentioned in the overview of the TMO scheme, the designer of each RT object can provide a guarantee of the timely service capabilities of the object by indicating the execution time-window for every output produced by each service method in the specification of the service method advertised to the designers of potential client objects. Actually the execution time-window associated with every output form every TT method is also a part of the guarantee.

An output action of a service method ma be one of the following

(1) An updating of a portion of the ODS;
(2) Sending a message to either another RT object (which may or may not be the client) or a device shared by multiple objects;
(3) Placing a reservation into the reservation queue for a certain TT method that will in turn take its own output actions.

The specification of each service method which is provided to the designers of potential client RT objects, must contain at least the following:

(1) An input specification that consists of
  (1a) the types of input parameters that the server object can accept and
  (1b) the maximum request acceptance rate, i.e., the maximum rate at which the server object can receive service requests form client objects;
(2) An output specification that indicates the maximum delay (not the exact output time) and the nature of the output value for every output produced by the service method.

If service requests form client object arrive at a server object at a rate exceeding the maximum acceptance rate indicated in the input specification for the server object, then the server may return exception signals to the client objects. The system designer who checks an interconnection of RT objects can prevent such "overflow" occurrences through a careful analysis. The designer should ensure that the aggregate arrival rate of service request at each server object does not exceed the maximum acceptance rate during any period of system operation. In order to satisfy greater service demands presented by the client objects, the system designer can increase the number of server objects or use more powerful execution engines in running server objects.

Before determining the maximum delay specification, the server object designer must consider the following.

(1) The worst-case delay form the arrival of a service request form a client object to the initiation of the corresponding service method by the server object;
(2) The worst-case execution time for the service method from its initiation to each of its output actions.

On the other hand, a client RT object imposes a deadline on the cooperating distributed object execution engines for producing the intended computational effects including the execution of the called service method and the arrival of the return results at the client object If this deadline is violated, the execution engine for the client object invokes an appropriate exception handling function.

The specifications of the TT methods which may be executed on requests from the service method must also be provided to the designers of the client objects which may call the service method. The specification of such a TT method must contain at least the time triggering specification and the output specification. There is no input specification. The output specification indicates, for every output expected from the execution of the TT method, the exact time at or by which it will be produced and the nature of every value carried in the output action.

VI. MULTI-LEVEL AND MULTI-STEP DESIGN FOR REAL TIME COMMUNICATION PROGRAM

First, the system engineering team describes the application environment as the TMO Mini-Theater in Fig. 3, without the components enclosed by square brackets. The components in brackets describes sensors (such as radar) which do not yet exist because the system engineering team has not decided which types to use.

The information kept in Mini-Theater is a composition of the information kept in all the state descriptors within its object data store. Here the object data store basically consists of the state descriptors for the following three environment components:

- Flying Airplane Group Container information (Environment)
- Flying Object Tracking Information(Reporter)
- Mini-Theater Space(Sky and Land)

corresponding to each of these state descriptors of environment components is a spontaneous method that periodically updates the state descriptor. Conceptually, spontaneous methods in Mini-Theater TMO are activated.

Mini-Theater

Access Capability (to other TMO's) None

Object Data Store

Mini-Theater Space (=Sky+Land Space)
Flying Airplane Group Container
Information(=Environment)
Flying Object Tracking information(=Reporter)
[Radar1 on Land]
[Radar2 on Land]

SpM "Update the state descriptors in ODS"
Update the state of Target in Land
[Update the state of Radar1 on Land]
[Update the state of Radar2 on Land]
[Update the state of Flying Airplane Group Container]
[Update the state of Reporter on Land]

SvM
Receive Flying Airplane Information From FAGC
Receive Request From Radars
Receive Flying Airplane Information From ASPACE
Receive Reporter From Radars

Fig. 3 High-level specification of the Mini-Theater TMO.

continuously and each of their executions is completed instantly. Spontaneous methods thus represent continuous state changes that occur naturally in the environment components. Multiple spontaneous methods activated simultaneously can be used to precisely represent the natural parallelism that exists among environment components.

The state descriptor for the theater space not only provides geographical information about the theater but also maintains the position of every moving component in the Mini-Theater. This information is used to determine the occurrences of collisions among components and to recognize the departure of any component from the Mini-Theater.

The Mini-Theater object is more than a mere description of the application environment; it is also a simulation model. To support simulation, the designers choose an activation frequency for each spontaneous method such that it can be supported by an object execution engine. The behavior of the environment can be simulated. This practical simulation is of course less accurate than the unexecutable description based on continuous activation of spontaneous methods. In general, the accuracy of a TMO-structured simulation is a function of the chosen activation frequencies of spontaneous methods.

Next the system engineering team decides which sensors to deploy. Sensors include two radars located on land. Once this is done, Mini-Theater can be expanded to incorporate all the components enclosed by square brackets in Fig. 3. The object data store now contains the selected sensors. The two radars loaded on Reporter are described in the state descriptor for the Reporter.

Now the system engineering team should also decide how to deploy the computer-based control system in the Mini-Theater.

Control Computer System In Reporter

Access Capability (to other TMOs)
Radar (Accept_spot_check_request)

Object Data Store
Radar data received, Flying airplane tracking information

SpM
SpM1 Radar Data Processing Step
- "Process all the radar data received since the last processing cycle, update the flying object tracks"

AAC : **for** T = **from TMO_START** + WARMUP_DELAY_SECS
to TMO_START + SYSTEM_LIFE_HOURS **every** PERIOD
start-during (T, T + START_WINDOW) **finish-by** T + DEADLINE
InputSpec : Radar data received in the object data store
OutputSpec : <deadline : xxx msec> Reflect changes onto the object data store, i.e., Radar data received, Flying airplane tracking info.
<deadline : xxy msec> Send spot-check radar requests to Radar if ···;

SvM
SvM1 Receive_from_Radar_on_Land (pos_list)
< **Accept-with-Delay_Bound-of** ACCEPTANCE_DEADLINE
under MAX_REQUEST_RATE **finish-within** EXECUTION_TIME_LIMIT>
- "Receive from Radar_on_Land the information on all recent detections."

InitiationCond : Other SvM1 invocations are not in place.
InputSpec : pos_list = array of (return_type (=scan_search/spot_check), position, time, predicted_time)
OutputSpec : <deadline : yyy msec> Deposit the radar data received in the object data store
SvM2 Accept Advice from ··· < **Accept-via**-··· >

Fig. 4 Intermediate Specification of the control computer system for Command Post.

The functions of the control system will be determined by the control theory logic adopted. In this experimental development, we deployed one control system such as Reporter.

The Reporter contains a control system. Initially, the system engineers proceed each control computer system out of Reporter and generate single TMO specification, as shown in Fig. 4. The specification in Fig. 4 shows a more complete specification structure than shown in Fig. 3. It has the autonomous activation condition for the spontaneous method, the input and output specifications for both the spontaneous and the service methods, and the initiation condition for the service method.

The Reporter contains a control system. Initially, the system engineers proceed each control computer system out of Reporter and generate single TMO specification, as shown in Fig. 4. The specification in Fig. 4 shows a more complete specification structure than shown in Fig. 3. It has the autonomous activation condition for the spontaneous method, the input and output specifications for both the spontaneous and the service methods, and the initiation condition for the service method.

•The input specification for a method describes the actions of picking data during the execution of the method such as receiving the data coming from the external client in the form of call parameters, picking data from the object data store, or picking data from the input devices.

•The output specification for a method describes the action of sending data to other TMOs, sending data to the output devices, and depositing data into the object data store.

•The initiation condition for the service method describes when the service method execution can be initiated after being called by a client. It is in a sense a concurrency specification.

Now Mini-Theater is a network of three objects. The system engineering team is now ready to give the computer engineering team the specification structured in the form of three TMOs, plus an overall specification of the type. Embed one control computer system in the Reporter such that the computer system follows the chosen control theory logic to control the chosen sensors such as radars.

Radar Data Queue

Access Capability (to other TMOs)
Flying Airplane Tracking Information (Receive_new_info_from_Radar_Data_Queue)

Object Data Store

Scan_Search_Result : MVD : x msec : list of {current_position, current_time}
Spot_Check_Predicted : MVD : y msec : list of {object_ID, predicted_position, predicted_time}
spot_Check_Result : MVD : z msec : list of {current_position, current_time, predicted_time}

SpM

SpM1 Send_New_Data_To_Flying_Airplane_Tracking_Information
- For each spot-check result in Data Set [Spot_Check_Result], match with predicted spot checks in Data set [Spot_Check_Predicted] by comparing the values of predicted_time.
  - If there are ones that match,
    - Construct a message containing the object_ID from Data Set [Spot_Check_Predicted] and the current_position and current_time from Data Set [Spot_Check_Result]
    - Send the message to Flying_Airplane Tracking_Information (via SvM request).
- For each scan search results in Data Set [Scan_search_Result], construct a message containing current_position and current_time of the objects.
  - Send the message to Flying_Airplane_Tracking_Information (via SvM request).

AAC : **for** T = **from** TMO_START + WARMUP_DELAY_SECS **to** TMO_START + SYSTEM_LIFE_HOURS **every** PERIOD **start-during** (T, T + START_WINDOW) **finish-by** T + DEADLINE
InputSpec : Scan_Search_Result, ····· in the object data store
OutputSpec : <deadline : xxx msec>Send list of (object_ID, position, time) to ····.

SvM

SvM1 <**Accept-via-Service_Request_channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE **under** MAX_REQUEST_RATE **finish-within** EXECUTION_TIME_LIMIT>
Receive_From_Radar_on_Land (pos_list)
- Update Data Set [Scan_search_Result] or [Spot_check_Result] according to the value of detection_type.

InitiationCond : Other SvM1 invocations are not in place.
InputSpec : pos_list = array of (return_type (=scan_search/spot_check), position, time, predicted_time)
OutputSpec : <deadline : yyy msec> *Deposit the radar data received* ·····
SvM2 <**Accept-via-**····>Receive_From_Flying_Airplane_Tracking_information (spot_list)

Fig. 5 Detailed design specification for Radar TMO.

To outline the detailed design, we will consider only the Reporter control computer system. To design this system, the computer engineering team initially produces a single TMO with an object data store comprising two major data structures:

•Radar Data Queue(RDQ), which contains radar data received; and

•Flying Airplane Group Container Tracking (FAGCT) information, which contains information needed for tracking flying airplane objects.

Some of the radar data coming into RDQ TMO happens as a result of spot-check requests generated by FAGCT TMO. To determine where to send the data, RDQ often references recent spot-check requests generated by FAGCT. To support this, FAGC sends a copy of each radar request to RDQ.

Fig. 5. shows the detailed design specification of RDQ, as would be generated by the computer engineering team. SvM1 receives information of flying airplane object in the Mini-Theater from radar and SvM2 receives copies of spot-check radar requests from FAGCT. SpM1 periodically sends radar data along with the ID numbers of the requests to FAGC.

FAGCT analyzes the radar return data and determines if the detected flying airplane object is dangerous. If it is not, FAGC simply tracks it for a short time and then forgets about it. Major FAGCT computations are handled by spontaneous methods, whereas service methods are designed mainly to receive information and deposit it into appropriate object data store segments.

In the real time simulation techniques based on TMO object modeling, we have observed several advantages to the TMO structuring scheme. TMO object modeling has a strong traceability between requirement specification and design, cost-effective high-coverage validation, autonomous subsystems, easy maintenance and flexible framework for requirement specification.

VII. CONCLUSION

Deadline handling is a fundamental part of real-time computing. This paper has proposed a general broadly applicable framework for systematic deadline handling in RT distributed objects. A prototype implementation of the basic middleware support for the proposed deadline handling scheme has been completed recently.

We also believe that using this scheme for the uniform, integrated design of complex real time systems and their application environment simulators offers great potential in significantly reducing the development costs and increasing the dependability of the real time systems. Also, the goal of the TMO structuring scheme, is to realize RT computing in a general manner not alienating the main-stream computing industry and yet enabling the system engineer to confidently produce certifiable real time simulator for safety-critical applications.

Although the potential of the TMO scheme has been amply demonstrated, much further research efforts are needed to make the TMO structuring technology easily accessible to common practitioners. Further development of TMO support middleware, especially those running on new-generation RT kernels and multiprocessor hardware, is a sensible topic for future research. Tools assisting the TMO designer in the process of determining the response time to be guaranteed are among the most important research topics

REFERENCES

[1] A. Attoui and M. Schneider, "An object-oriented model for parallel and reactive systems", *Proc. IEEE CS 12th Real-Time Systems Symp.*, pp. 84-93, 1991.

[2] K. H. Kim et al., "A timeliness-guaranteed Kernel model DREAM kernel and implementation techniques", *Proc. 1995 Intl Workshop on Real-Time Computing Systems and Applications (RTCSA 95)*, Tokyo, Japan, pp. 80-87.Oct. 1995.

[3] K. H. Kim, C. Nguyen, and C. Park, "Real-time simulation techniques based on the RTO.k object modeling", *Proc. COMPSAC 96 (IEEE CS Software & Applications Conf.)*, Seoul, Korea, pp. 176-183, August 1996.

[4] K. H. Kim and C. Subbaraman, "Fault-tolerant real-time objects", *Commun. ACM* pp.75-82. 1997.

[5] K. H. Kim, C. Subbaraman, and L. Bacellar, "Support for RTO.k Object Structured Programming in C++", *Control Engineering Practice* 5 pp. 983-991, 1997.

[6] K. H. Kim, "Object Structures for Real-Time Systems and Simulators", *IEEE Computer 30*, pp. 62-70, 1997.

[7] H. Kopetz and K. H. Kim, "Temporal uncertainties in interactions among real-time objects", *Proc. IEEE CS 9th Symp. On Reliable Distributed Systems*, pp. 165-174, Oct. 1990.

[8] J. C. Laprie, "Dependability: A Unifying Concept for Reliable, Safe, secure Computing", *in Information Processing*, ed. J. van Leeuwen, pp. 585-593,1992.

[9] C. W. Mercer and H. Tokuda, "The ARTS real-time object model", *Proc. IEEE CS 11th Real-Time Systems Symp.*, pp. 2-10, 1990.

[10] Kim, K.H.,"Real time Object-Oriented Distributed Software Engineering and the TMO scheme", *Int'l Jour. of Software Engineering & Knowledge Engineering*, Vol. No.2, pp. 251-276, April 1999.

Gwang-Jun Kim, was born in Kwangju, south Korea, on July 30, 1964. He received the B.E, M.E and Ph. D. degrees in computer engineering from Chosun University in 1993, 1995 and 2000, respectively. He joined the department of computer engineering, Yosu National University, in 2003. Since 2003, he has been a full-time lecturer in the division of computer engineering at Yosu National University. During 2000-2001, he was a researcher in the department of electrical and computer engineering at university of California, Irvine. His current research interests lie in the area adaptive signal processing, real-time communication, TCP/IP network, spread spectrum system, ATM network and various kinds of communication systems. He is a member of ISS of Korea, ICS of Korea, ITE of Korea and IMICS of Korea

Chun-Suk Kim, was born in Yous , south Korea, on October 03, 1954. He received the B.E degree in electrical engineerging from the University of Kwangwoon in 1980, the M.E degree in electrical engineering from the University of Gunkook in 1982, and the Ph. D. degree in electrical engineering from the University of Keongnam in 1998. He joined the Department of Computer Engineering, Yosu National University, in 1980 and became an Assistant Professor, Associate Professor in 1982 and 1989, respectively. Since 1993, he has been a Professor in the Division of Electronic Communication Engineering at Yosu National University. His research interests lie in the area of digital signal processing, radio communication, information theory and various kinds of communication systems. He is a member of ISS of Korea, ICS of Korea, ITE of Korea and IMICS of Korea

Yong-Gin Kim, was born in Kwangju, south Korea, on January 23, 1961. He received the B.E and M.E degree in electrical engineering from the University of Chonnam in 1983 and 1985, respectively. He is currently completing the Ph. D. degree in electrical engineering at University of Chonnam, since 1998. In April 1985, he joined the faculty of LGcaltex company where he is currently an senior researcher. His research interests lie in the area of digital signal processing, electric power communication, digital communication theory, error control coding and various kinds of communication systems. He is a member of ISS of Korea, ICS of Korea, ITE of Korea and IMICS of Korea

Chan-Ho Yoon, was born in Kwangju, south Korea, on July 07, 1975. He received the B.E degree in computer engineering from the University of Honam in 1997, the M.E degree in computer engineering from the University of Chosun in 2000. He is currently completing the Ph. D. degree in computer engineering at University of Chosun, since 2000. His research interests lie in the area of digital signal processing, real-time communication, ATM network, multimedia communication and various kinds of communication systems. He is a member of ISS of Korea, ICS of Korea, ITE of Korea and IMICS of Korea

Moon-Hwan Kim, was born in Jeju, south Korea, on January 09, 1961. He received the B.E degree in computer engineering from Korea National Open University in 1988, the M.E degree in computer engineering from the University of Chosun in 1991. He is currently completing the Ph. D. degree in computer engineering at University of Chosun, since 2001. His research interests lie in the area of digital signal processing, switching network operation, real-time communication, ATM network, multimedia communication and various kinds of communication systems. He is a member of ISS of Korea, ICS of Korea, ITE of Korea and IMICS of Korea