Deadline Handling in Real-Time Distributed Object Oriented Programming of TMO

Hee-Chul Kim* · Sang-Dong Na**

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.

Keyword

Real-Time(RT) Operating System Kernel, Deadline Handing, Guaranteeing time, 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. 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[1,2,3,4].

Another issue of growing importance is to provide in the future an order-of-magniture higher degree of assurance on the reliability of distributed time triggered simulation products than what is provided today[5,6]. To require the system engineer to produce design-time guarantees for timely service capabilities of various subsystems which will take the form of objects in OO system designs.

*케이티솔루션스

*조선대학교 컴퓨터공학과

접수일자 : 2002. 10. 23

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.

II. General framework for systematic deadlin handling

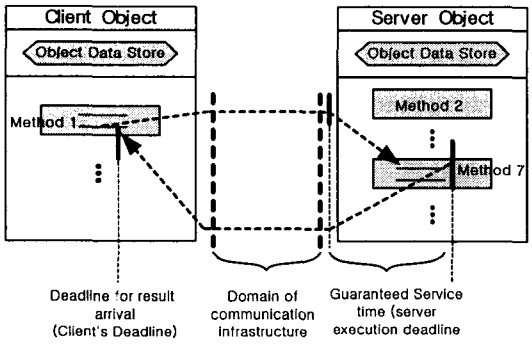


Fig. 1. Client's deadline vs Server's guaranteed service time

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, Method 1, 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. 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.

III. An overview of the TMO scheme

The TMO programming scheme is a general style component programming scheme and supports design of all types of components including distributable hard-RT object and distributable non-RT objects within one general structure. TMOs are devised to contain only high-level intuitive and yet precise expressions of timing requirements. No specification of timing in terms other than start-windows and completion deadlines for program units and time-windows for output actions is required[4,6,7,8].

The TMO scheme is aimed for enabling a great reduction of the designer's effort 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.

The basic structure of the TMO model consists of four parts as follows[6]:

TMO= 〈ODS-sec, EAC-sec, SpM-sec, SvM-sec〉

Where

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

EAC-sec = environment access-capability section: list of TMO-name. SvM-names programmable communication channels, and I/O devices;

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

SvM-sec = Service-method section: list of service-methods.

The TMO model is a syntactically minor and semantically powerful extension of the conventional object model. Significant extensions are summarized below and the second and third are the most unique extensions.

(a) Distributed computing component

The TMO is a distributed computing component and thus TMOs distributed over multiple modes 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 or 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 or SpMs) which are clearly separated form 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 SpM's.

(c) Basic concurrency constraint(BCC)

This rule prevents potential conflicts between SpM's and SvM's and reduces the designers efforts in guaranteeing timely service capabilities of TMO's. Basically, activation of an SvM triggered by a message form an external client is allowed only when potentially conflicting SpM executions are not in place. An SvM is allowed to execute only if no SpM that accesses the same object data store segments(ODSS's) to be accessed by this SvM has an execution

time-window that will overlap with the execution time-window of this SvM.

(d) Guaranteed completion time and deadline

As in other RT object models, the TMO incorporates deadlines and it does so in the most general form. Basically, for output actions and completion of a method of a TMO, the designer guarantees and advertises execution time-window bounded by start times and completion times. Triggering times for SpM's must be fully specified as constants during the design time. Those RT constants appears 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 30 min start-during (t, t+5 min) finished-by t+10 min. 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 or SpM within the same TMO requests future executions of a specific SpM. TMO's interact via calls by client objects for service methods in server objects. The caller maybe an SpM or an SvM in the client object. The designer of each TMO provides a guarantee of timely service capabilities of the object.

He/she does so by indicating the guaranteed execution time-window for every output produced by each SvM as well as by each SpM executed on requests from the SvM and the guaranteed completion time for the SvM in the specification of the SvM. Such specification of each SvM is advertised to the designers of potential clients objects. Before determining the time-window specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) 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. Models and prototype implementations of the effective operating system(OS) support and the friendly application programmer interface(API) have been developed.

The TMO model is effective not only in the multiple-level abstraction of RT (computer) control systems under design but also in the accurate representation and simulation of the application environments. In fact, it enables uniform structuring of control computer systems and application environment simulators and this presents considerable potential benefits to the system engineers.

IV. RT object structuring tool and the TMO approach

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.

4.1. 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.

4.2. 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 a 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.

4.3. Separation of the absolute time domain fro 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. Simulation with the TMO structuring

The attractive basic design style facilitated by the TMO structuring is to produce a network of TMO's meeting the application requirements in a top-down multi-step fashion. For each environment object represented by a state descriptor in the Theater TMO, there is a spontaneous method (SpM) for periodically updating the state descriptor. Conceptually the SpM's in the Theater TMO are activated continuously and each of their executions is completed instantly. The SpM's can then represent continuous state changes that occur naturally in the environment objects. The natural parallelism that exists among the environment objects can also be precisely represented by use of multiple SpM's which may be activated simultaneously.

The service methods (SvM's) in the Theater TMO are provided as an interface for the clients outside the Theater. The only conceivable clients

here are the enemy which send RVs into the Theater to enter the Theater. Entry of an RV into the Theater is represented by and enemys call for the SvM Accept RV. Both the enemy and the external forces are represented by a TMO called the Alien TMO.

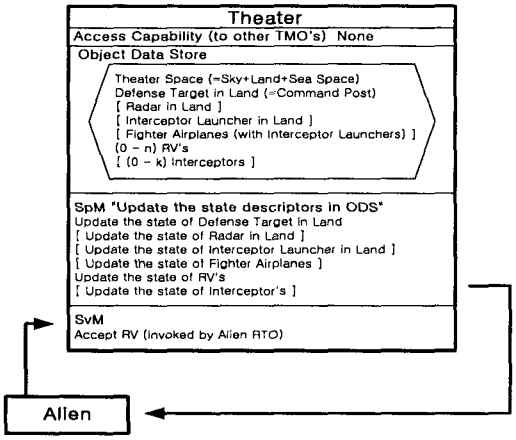


Fig. 2. High-level specification of the Theater TMO

So far, the Theater TMO in Fig. 2 has been interpreted as a mere description of the application environment. However, if the activation frequency of each SpM is chosen such that it can be supported by an object execution engine, then the resulting Theater TMO becomes a simulation model. The behavior of the application environment is represented by this simulation model somewhat less accurately than by the earlier description model based on continuous activation of SpM's. In general, the accuracy of a TMO structured simulation is a function of the chosen activation frequencies of SpM's. Upon receiving the customers order, the system engineer will first decide on the set of sensors and actuators to be deployed in the Theater. After the set of sensors and actuators is determined, the Theater TMO in Fig. 2 is expanded to incorporate all the components enclosed by square brackets. The ODS now contains the selected sensors (e.g., Radar in Land) and actuators (i.e., Interceptor Launcher in Land with Interceptors). The radar and another interceptor launcher on the fighter airplane are not shown in the ODS of the Theater TMO but these environment objects are described in the corresponding parts of the state descriptors for the command ship and the fighter airplane, respectively.

The Theater space component in the ODS of the Theater TMO not only provides geographical information about the Theater but also maintains the position information of every moving object in the Theater. This information is used to determine the occurrences of collisions among objects and to recognize the departure of any object from the Theater space to the outside.

As the system engineer refines the single TMO representation of the Theater, a component in the ODS of the Theater TMO may be taken out of the Theater TMO to form a new TMO. such separation of the command post from the Theater TMO. Therefore, the Command Post TMO and the Command Ship TMO are born. When the new TMO's are created, the SvM's that serve as front-end interfaces of those new TMO's and the call links from the earlier born TMO's to the new TMO's should also be created. As a result, the Theater TMO becomes a network of three TMO's.

The two new TMO's may describe or simulate the command post and the command ship more accurately than the Theater TMO in Fig. 2 did.

Now 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

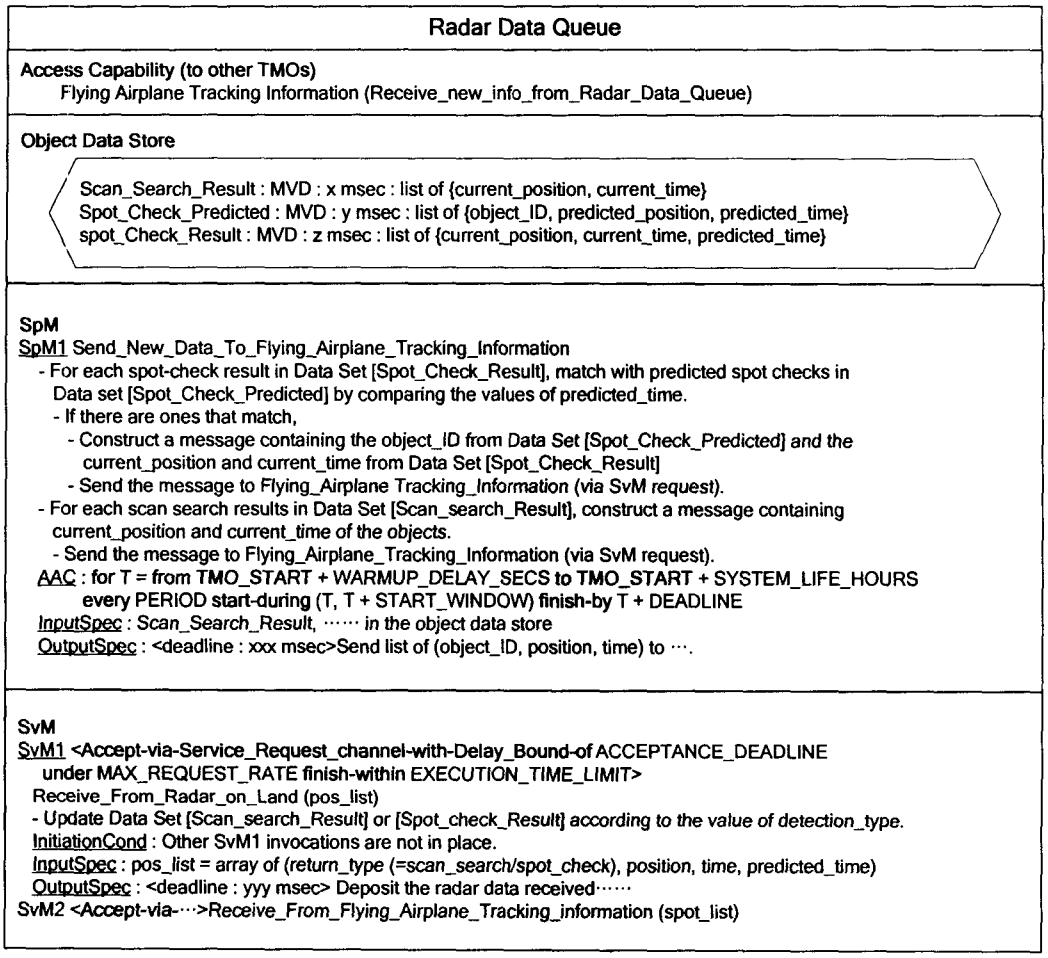


Figure3. Detailed design specification for Radar TMO.

chosen sensors such as radars.

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.

Figure 3 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.

VI. RT object structuring tool and the TMO approach

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. However, the cases where advanced RT fault tolerance techniques such as those for active replication of TMO method executions are used, have not yet been dealt with and remain a subject for future study. Systematic deadline handling is an area where much more experimental research is needed.

References

[1] 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.

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

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

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

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

[6] 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.

[7] K. H. Kim., APIs Enabling High-Level Real-Time Distributed Object Programming, to appear in IEEE Computer, June 2000.

[8] G.J.K, S.D.Na and C.S.Bae, Time Service Guarantee in Real-Time Distributed Object Oriented Programming of TMO, Proc. ICIM01, pp.215-219, Nov., 2001.

저자소개

김희철(Hee-Chul Kim)

1988년 광주대학교 전자계산학과 졸업(이학사)

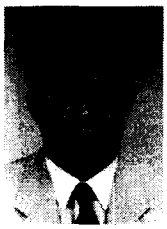
1990년 조선대학교 대학원 졸업(공학석사)

2000년 조선대학교 대학원 컴퓨터공학과(박사수료)

1982년~1985년 보병 73훈련단 206연대 통신과장

1988년~현재 케이티솔루션스 근무

※관심분야: 실시간 통신, 디지털 통신망, 데이터 및 이동통신, TMO, VoIP.



나상동(Sang-dong Ra)

1968년 조선대학교 전기공학과 졸업(공학사)

1980년 건국대학교 대학원 졸업(공학석사)

1995년 원광대학교 대학원 졸업(공학박사)

1995년~1996년 Dept. of Electrical & Computer Eng. Univ. of California Irvine 연구교수

1998년 조선대학교 전자계산소 소장 역임

1973년~현재 조선대학교 컴퓨터공학부 교수

2001년~2002년 Dept. of Electrical & Computer Eng. Univ. of California Irvine 연구교수

※관심분야: 실시간 통신, 디지털 통신망, 데이터 및 이동통신, TMO, 적응 신호처리 등임.