Real Time simulation programming in Object Oriented Distributed Computing Systems

Yong-Geun Bae* · Dal-Bok Chin**

객체지향 분산 컴퓨팅 시스템에서 실시간 시뮬레이션 프로그래밍

배용근* · 진달복**

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 Real Time Simulation Programing(RTSP) structuring scheme was formulated with the goal of instigating a quantum productivity jump in the design of distributed time triggered simulation. The RTSP 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 tc allow system engineers to confidently produce certifiable distributed time triggered simulation for safety-critical applications. The RTSP 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.

요 약

실시간 객체지향 분산 컴퓨팅은 객체 네트워크 형태에서 분산된 컴퓨터 시스템 구조와 관련된 실시간 분산 컴퓨팅의 한가지 형태이다. 최근에 실시간 응용분야에 적합한 기존의 객체지향 시스템 구조를 확장한 몇 가지의 구조가 제안되었다. 실시간 시뮬레이션 프로그램의 하나인 시간 및 메시지 트리거 객체지향 프로그램밍이 분산된 시간 트리거 시뮬레이션으로 설계될 수 있으며, 일반적이고 보편적인 설계 타입으로서 사용되고, 하나의 실시간 시뮬레이션 패러다임으로 제안하였다. 실시간 객체지향 프로그램밍은 안전을 중요시하게 여기는 응용분야에 적용할 수 있으며, 실시간 운영체제 시스템 커널로서 객체지향 프로그래밍 언어인 비쥬얼 C++언어로 작성되었다. 응용 시스템에서 실시간 서비스를 보장하기 위한 설계자들의 노력을 현저하게 줄일 수 있는 장점을 가지고.있다.

키워드

Real-Time(RT) Operating System Kernel, Guaranteeing time, Real-Time Simulation Programing(RTSP)

* 조선대학교 전자정보공과대학 컴퓨터공학부
** 원광대학교 공과대학

접수일자 : 2002. 4. 12

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.

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.

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[1][2][3][4][5]

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 frame work 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.

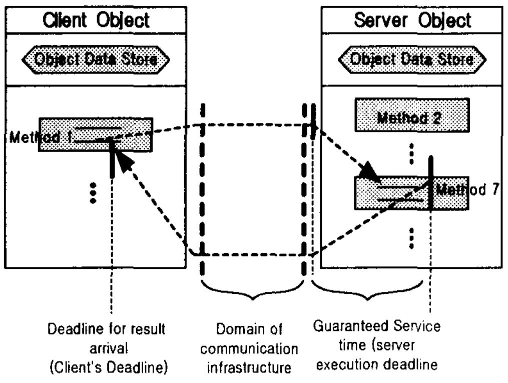

Client Object
Object Data Store
Method 1
Server Object
Object Data Store
Method 2
Method 7
Deadline for result arrival (Client's Deadline)
Domain of communication infrastructure
Guaranteed Service time (server execution deadline

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 [6][7][9].

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 source 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 Real-Time Simulation Programing scheme

The RTSP 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[9][11].

RTSPs 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.

The RTSP 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 RTSP incorporates several rules for execution of its components that make the analysis of the worst case time behavior of RTSPs to be systematic and relatively easy while not reducing the programming power in any way.

The basic structure of the RTSP model consists of four parts as follows:

RTSP = 〈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 RTSP-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 RTSP 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 RTSP is a distributed computing component and thus RTSPs 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 RTSP 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 RTSP'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 RTSP incorporates deadlines and it does so in the most general form. Basically, for output actions and completion of a method of a RTSP, 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 RTSP requests future executions of a specific SpM.

RTSP'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 RTSP 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 RTSP 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. Interaction among RT objects and RT message communication

4.1 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 RTSP scheme supports the following two basic types of calls to service methods in the server RTSP.

(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.

4.2 Client-transfer call

Even though its needs were initially recognized in the context of the RTSP scheme, it is of fundamental nature and may be useful in almost all types of RT systems, Basically, an SvM in a RTSP 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 SpMs 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 SvMs, 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 RTSP. The syntactic structure for such a client-transfer call for an external SvM is about the same, i.e.,

ClientTransferCall(Obj_name.SvM_name,paramete rs)

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.

4.3 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.1. For example, logical multicast channels, LMC1 and LMC2, can be declared as data members of each of the three remotely cooperating RT objects, RTSP1, RTSP2, and RTSP3, 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 RTSP1 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 RTSP2 and RTSP3 can pick up the messages that came over LMC1 into the ODSs of their host objects. In many applications, this interaction mode leads to better efficiency than the interaction mode based on remote method invocations does.

V. RT Object Structuring Tool and the RTSP Approach

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

5.1. 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 RTSP scheme provides these facilities.

5.2 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 RTSP 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 RTSP structure that prevents the incorporation of TT initiation into other computation units. Such an extension just requires construction of RTSP 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.

5.3. 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 RTSP 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.

Ⅵ. 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. However, the cases where advanced RT fault tolerance techniques such as those for active replication of RTSP 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, Temporal 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] 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

[8] K. H. Kim et al., A timeliness-guaranteed Kernel model DREAM kernel and implementation techniques, Proc. 1995 Intl Workshop

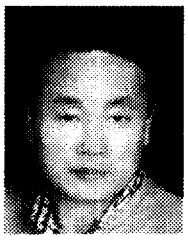
on Real-Time Computing Systems and Applications (RTCSA 95), Tokyo, Japan, pp. 80-87.Oct. 1995

[9] 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

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

[11] 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

저 자 소 개



배용근(Young-Geun Bae)
1984년 조선대학교 공과대학 컴퓨터공학과 졸업
현재 조선대학교 전자정보공과대학 컴퓨터공학부 부교수

※관심분야 : 마이크로 프로세서 응용, 프로그래밍 언어

진달복(Dal-Bok Chin)
1972년 조선대학교 대학원졸(공학석사)
1985년 전남대학교 대학원졸(공학박사)
1970년~1982년 조선대학교 공과대학 조교수
1982년~현재 원광대학교 공과대학 교수
1987년~1991년 원광대학교 공과대학장
1991년~1993년 원광대학교 기획처장
※관심분야 : 마이크로프로세서 응용, 음성처리