Distributed Real Time Simulation Programming with Time and Message Object Oriented in Computer Network Systems

Regular Member *Sang-Dong Ra, *Ha-sun Na, *Moon-hwan Kim

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.Start after striking space key 2 times.

Keywords: Real-Time(RT) Operating System Kernel, Guaranteeing time, Time-triggered Message-triggered Object (TMO)

I. INTRODUTION

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.

The major factor that has discouraged any attempt to do this has been the use of software

*Computer Engineering of Chosun University(sdna@mail.chosun.ac.kr), **Dept. of Electrical and Computer Engineering Graduate School University of Colorado at Boulder

논문번호 : #030451-1014, 접수일자 : 2003년 10월 1일

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

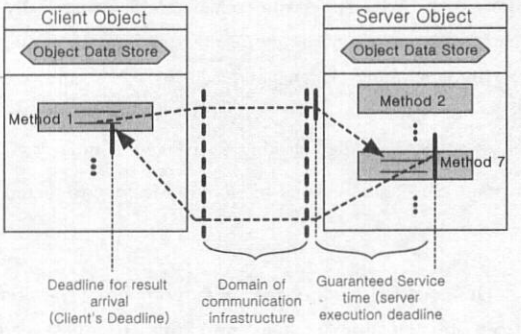

Fig.1. Client's deadline vs Server's guaranteed service time

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, 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.

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. 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:

TMO="qODS-sec, EAC-sec, SpM-sec, SvM-sec"r

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.

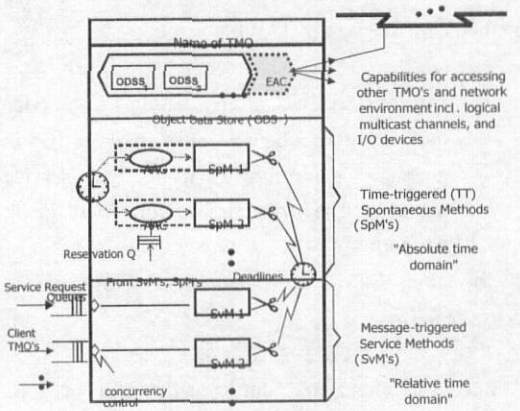

Fig. 2. Structure of the TMO.

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 TMO's 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 SpM's) which are clearly separated form the conventional service methods(SvM's).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 SvMand 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. Interaction among RT objects and RT message communicati

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 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.

4.2 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 ina 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 thecase 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.

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, 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.

4.4 Deadline specification and service 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.

V. 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 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] 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.

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

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

[4] K. H. Kim., "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

[5] 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

[6] K.H.Kim and Liu, J. "Deadline Handling in Real-time Distributed Objects", *Proc. ISORC 2000, Newport Beach, CA*, pp.7-15, March 2000

[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.Ra and C.S.Bae, "Time Service Guarantee in Real-Time Distributed Object Oriented Programming of TMO", *Proc.ICIM'01*, pp.215-219, Nov.2001

[9] C.J.Jeong and S.D.Ra, "High Level Object Oriented Real-Time Simulation Programming and Time-triggered Message-triggered Object(TMO) Scheme", The KIMICS, Journal, VOL.6, No.6, Oct 2002

[10] H.C.Kim and S.D.Ra, "Deadline Handling in Real-Time Distributed Object Oriented Programming of TMO", The KIMICS, Journal, VOL.6, No.6, Oct 2002

[11] S.H.Song and S.D.Ra, " High Level Object Oriented Real-Time Simulation Programming and TMO Scheme", The KIPS Transactions : PartA, Journal, VOL.10-A,No3, August 2003

Sang-dong Ra

Was born in Naju, South Korea, on February 9, 1945. He received the B.E degree in electrical engineering from the University of Chosun in 1968, the M.E degree in electrical engineering from the University of Gunkook in 1980, and the Ph. D. degree in electriacl engineering from the University of Wookwang in 1995. He joined the Department of Computer Engineering, Chosun university, in 1983 and became an Assistant Professor, A associate Professor in 1985 and 1990, respectively. Since 1994, he has been a Professor in the Division of Computer Engineering at Chosun University. Currently, he is a Chairman of Department of Computer Engineering at Chosun university. During 1995-1996, 2000-2001, he was a Visiting Scientist in the Department of Electrical and Computer Engineering at the University of California, Irvine. He is a member of System H/W Subcommittee, Expert Committee, Industrial Technology Development, Ministry of Trade and Industry. His current research interests lie in the areas of adaptive signal processing, information theory, data communication, spread spectrum systems and various kinds of communication systems. He is member of ISS of Korea, ICS of Korea, AS of Korea and ITE Korea. He is also a former member of the IEEE communication Society Board of Governors

Ha-Sun Na

Received the B.S. degree from Hankook Aviation Unversity in 1998, and the M.S. degree in electrical and computer engineering from the University of Colorado at Boulder in 2003. He participated in the Reconfigurable Aperture project supported by the Camp mode researcher center of the Univ. of Colorado and worked on a RF-MEMS switching device and a slot antenna in his graduate study. His interest area of research are RF communication. Bioelectromagnetics and real-time communication

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. He is R&D Center team manager KRTnet corporation in 2004. 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.