High Level Approach Programming in Real Time Distributed Network System

Chan-Joo Jeong[1], Gwang-Jun Kim[2], Joon Lee[3], Ki-Hwan Nam[4] and Chul-Soo Bae[5]
[1,2,3]School of Computer Engineering, Chosun University, Kwangju, Korea
Tel: +82-062-230-7757, Fax: +82-062-230-7381
[4,5] Dept. of Information & Communication Engineering, Kwandong University, Kangwon, Korea
Tel: +82-033-670-3411, Fax: +82-033-671-2118
e-mail : jeong-jcj@hanmail.net ,ban9628@hotmail.com, jlee@mail.chosun.ac.kr,
keelight@hanmail.net , baecs@mail.kwandong.ac.kr

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.

1. 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.

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.

2. Framework for 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. 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.

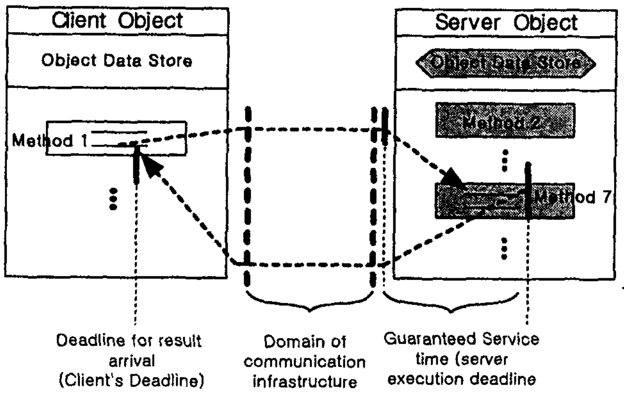
Client Object
Object Data Store
Method 1
Server Object
Object Data Store
Method 2
Method 7
Deadline for result arrival (Client's Deadline)
Domain of communication infrastructure
Guaranteed Service time (server execution deadline

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

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.

3. An overview of the TMO scheme

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. 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.

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.

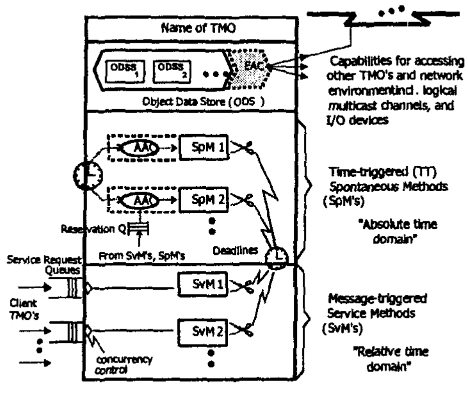

Fig. 1. Structure of the TMO.

(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.

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.

4. Multi-level Multi-step Design with the TMO Structuring

First, the system engineering team describes the application environment as the TMO Mini-Theater in Figure 2, 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:

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

Figure 2. High-level specification of the Mini Theater TMO.

- 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

```
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.
    InputSec : 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-…  .>
```

Figure 3. Intermediate Specification of the control computer system for Command Post.

periodically updates the state descriptor. Conceptually, spontaneous methods in Mini-Theater TMO are activated 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 Fiqure 2. 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 deside how to deploy the computer-based control system in the Mini-Theater. 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 Figure 3. The specification in Figure 3 shows a more complete specification structure than shown in Figure 2. 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.

5. Conclusion

We 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* 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