Object Oriented Real Time Distributed Programming and Time-triggered Message-triggered Object(TMO) Scheme

Gwang-Jun Kim*, Sang-Dong Ra*, Chul-Soo Bae** *Regular Members*

ABSTRACT

The object-oriented(OO) distributed real-time(RT) programming movement started in 1990's and is growing rapidly at this turn of the century. Distributed real-time simulation is a field in its infancy but it is bounded to receive steadily growing recognition for its importance and wide applicability. The scheme is called the distributed time-triggered simulation scheme which is conceptually simple and easy to use but widely applicable. A new generation object oriented(OO) RT programming scheme is called the time-triggered message triggered object(TMO) programming scheme and it is used to make specific illustrations of the issues. The TMO structuring scheme is a general-style components structuring scheme and supports design of all types of component including hard real time objects and non real time objects within one general structure.

Ⅰ. Introduction

Object oriented real time distributed computing is a rapidly growing branch of computer science and engineering. Its growth is fueled by the strong needs present in industry for the RT programing methods and tools which will bring about orders of magnitude improvement over the traditional RT programing practiced with low-level programing languages and styles.

RT simulator developments are under increasing demands[1,2,3,4]. For example, continuing advances in virtual reality application accompany increasing demands for more powerful RT simulation capabilities. Numerous other examples can also be found in the RT computing control field. Not only description but also simulation of application environments is often performed as integral steps of validating control computer system designs. RT simulators of application environments can often enable highly cost-effective testing of the control computer systems implemented. Such testing can be a lot cheaper than the testing performed in actual application environments while being much more effective than the testing based on non-RT simulators of environments.

The new generation OO RT programing scheme called the time-triggered message triggered object programing scheme[3,5,6]. In the course of developing a RT system engineering methodology based on this TMO programing scheme, a new approach to RT simulation which is conceptually simple and easy to use but widely applicable, has also been established.

In the next section, the motivations for pursuing the OO RT programing approach are reviewed and then in section 3, a brief overview is taken of the particular programing scheme. This programing scheme called the time-triggered message triggered object(TMO) programing scheme [1,2,3,6] is used on several occasions in the rest of this paper to make specific illustrations of the issues and potentials of OO RT programing.

Ⅱ. High-level approach to programming real-time distributed systems

As a concrete example of a high-level OO RT distributed programming approach that has been

* 조선대학교 컴퓨터공학부(ban9628@hotmail.com, sdna@mail.chosun.ac.kr)
** 관동대학교 정보통신공학부(baecs@mail.kwandong.ac.kr)
논문번호 : 020227-0509, 접수일자 : 2002년 5월 9일

based on the philosophy discussed in the preceding section, the time-triggered message-triggered object(TMO) programming scheme is briefly summarized in this section[2,3,4,5,6].

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

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

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

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

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

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

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

As depicted in Fig. 1 the basic TMO structure consists of four parts :

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

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

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

SvM-sec = service-method section.

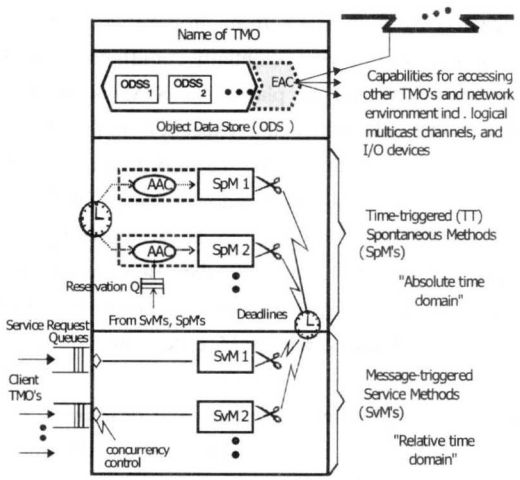

Fig. 1 Structure of the TMO

Major features are summarized below.

(a) Distributed computing component :

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

(b) Clear separation between two types of methods :

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

(c) Basic concurrency constraint (BCC) :

This rule prevents potential conflicts between SpMs and SvMs and reduces the designer's efforts in guaranteeing timely service capabilities of TMOs. Basically, activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place. An SvM is allowed to execute only when and execution time-window big enough for the SvM that does not overlap with the execution time-window of any SpM that accesses the same ODSSs to be accessed by the SvM, opens up. However, the BCC does not stand in the way of either concurrent SpM executions of concurrent SvM executions.

(d) Guaranteed completion time and deadline :

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

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

"for t = from 10am to 10:50am every 30min
start-during (t, t+5min) finish-by t+10min"
which has the same effect as
"start-during (10am, 10:05am)
finish-by 10:10am",
"start-during (10:30am, 10:35am)
finish-by 10:40am"

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

Ⅲ. Interaction among RT objects and RT message communication

1. Non-blocking call

An underlying designs philosophy of the RT OO distributed computing approaches is that every RT DCS will be designedin 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.

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 in a TMO may pass a client request to another SvM by using a client-transfer call. The latter SvM may again pass the client request to another SvM. This chaining sequence may repeat until the last SvM in the chain returns the results to the client. The main motivation behind such a client-transfer call stamp from BCC which require an execution of an SVM to be made only if a sufficiently large time-window between executions of SpM's potentially conflicting with the SvM opens up. Hence, in certain situations a highly complicated SvM may never be executed due to the lack of a wide enough time-window. One way to get around this problem is to divide the SvM into multiple smaller SvM's, SvM1,, SvMx. A client can then call each smaller SvM each time. Calling each smaller SvM incurs the communication overhead of transmitting a request to the smaller SvM and obtaining the results. Substantial

reduction of such communication overhead is the motivation behind an arrangement in which the client calls the first SvM and the latter passes the "service contract" with the client on to another SvM and so on until the last SvM of the chain returns the results to the client.

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

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

ClientTransferCall(SvM_name, parameters)

SvM_name identifies the SvM being called. The ID of the port or channel through which the current client of the caller SvM is prepared to receive return results is passed by the object execution engine on to 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.

3. Deadline specification and service time guarantee

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

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

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

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

(1) An input specification that consists of
(1a) the types of input parameters that the server object can accept and

(1b) the maximum request acceptance rate, i.e., the maximum rate at which the server object can receive service requests form client objects;

(2) An output specification that indicates the maximum delay (not the exact output time) and the nature of the output value for every output produced by the service method.

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

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

(1) The worst-case delay form the arrival of a service request form a client object to the initiation of the corresponding service method by the server object;

(2) The worst-case execution time for the service method from its initiation to each of its output actions.

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

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

Ⅳ. 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

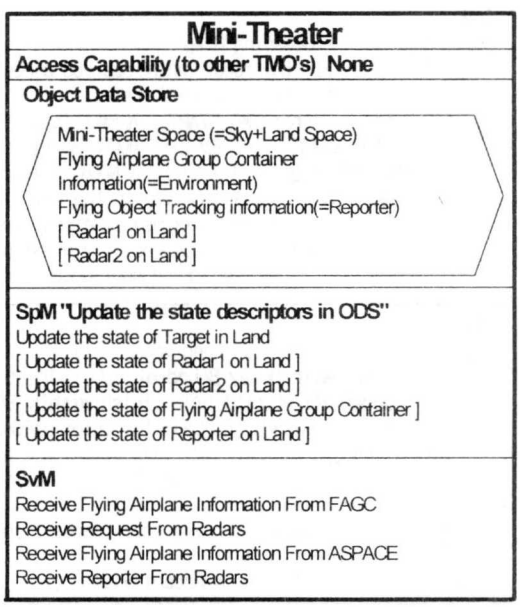

Fig. 2 High-level specification of the Mini Theater TMO.

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

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

Corresponding to each of these state descriptors of environment components is a spontaneous method that periodically updates the state descriptor.

Conceptually, spontaneous methods in Mini-Theater TMO are activated 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

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**… .>

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

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 Figure 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 decide 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. Embed one control computer system in the Reporter such that the computer system follows the chosen control theory logic to control the chosen sensors such as radars.

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 4 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

Radar Data Queue

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

Object Data Store

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

SpM

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

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

SvM

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

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

Fig. 4 Detailed design specification for Radar TMO

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.

V. Conclusion

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

김 광 준(Gwang-Jun Kim) 정회원

1993년 2월 : 조선대학교 컴퓨터공학과 졸업(공학사)
1995년 2월 : 조선대학교 대학원 컴퓨터공학과 졸업 (공학석사)
2000년 2월 : 조선대학교 대학원 컴퓨터공학과 졸업 (공학박사)
2000년~2001년 : Dept. of Electrical & Computer Eng. Univ. of California Irvine Postdoctoral
2001년~현재 : 베네테크(주) 수석연구원
<주관심 분야> ATM망, 실시간 통신 프로그래밍, 이동통신

나 상 동(Sang-Dong Ra) 정회원

1968년 2월 : 조선대학교 전기공학과 졸업(공학사)
1980년 2월 : 건국대학교 대학원 졸업(공학석사)
1995년 2월 : 원광대학교 대학원 졸업(공학박사)
1995년~1996년 : Dept. of Electr ical & 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
<주관심 분야> 실시간 통신, 디지털 통신망, 데이터 및 이동통신, TMO, 적응 신호처리 등임.

배 철 수(Chul-Soo Bae) 정회원

1979년 2월 : 명지대학교 전자공학과 졸업(공학사)
1981년 2월 : 명지대학교 대학원 전자공학과졸업 (공학석사)
1988년 8월 : 명지대학교 대학원 전자공학과졸업 (공학박사)
1991년~현재 : 관동대학교 정보통신공학부 교수.
1998년 3월 1일~2001년 5월 : 관동대학교 창업보육센터소장
1999년 3월 1일~2001년 5월 : 관동대학교 공과대학 학장
2001년 6월~2002년 10월 현재 : 관동대학교 평생교육원장
<주관심 분야> 디지털신호처리, 영상처리, 신경회로망 등임