

# High Level 객체 지향에서 실시간 시뮬레이션 프로그램과 TMO 설계

송 선 희<sup>†</sup> · 나 상 동<sup>††</sup>

## 요 약

객체 지향 분산 실시간 시스템은 1990년도에 시작되어 현재 빠르게 급성장하고 있는 공학의 한 분야이다. 분산 실시간 시뮬레이션은 실시간의 초기 분야로서 실시간 시뮬레이션의 중요함을 광범위 하게 나타냈으며 점진적으로 시뮬레이션이 크게 사용 할 수 있음을 TMO를 통해 보인다. 분산 시간-구동 시뮬레이션 구조도 개념적으로 간단하고, 사용하기 쉬우며, 또한 광범위하게 적용 및 응용이 가능성도 연구한다. 차세대 객체 지향 실시간 시뮬레이션 프로그래밍 구조에서 시간구동과 메시지 구동 객체지향(TMO) 프로그래밍 구조를 실시간 시뮬레이션 프로그래밍의 특별한 예시로서 사용하였다. 연구에서 제시한 TMO 구조도 일반적인 형태의 컴포넌트 구조로 이루어졌기 때문에 구조 내에서 실시간 객체 또는 비실시간 객체를 포함한 모든 형태의 컴포넌트 설계 등으로 지원함을 보인다.

## High Level Object Oriented Real-Time Simulation Programming and TMO Scheme

Sun-Hee Song<sup>†</sup> · Sang-Dong Ra<sup>††</sup>

### 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.

키워드 : 객체 지향(Object Oriented), 실시간 통신(Real-Time Communication), 컴퓨터 네트워크(Computer Network), TMO

### 1. 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 programming methods and tools which will bring about orders of magnitude improvement over the traditional RT programming practiced with low-level programming languages and styles.

RT simulator developments are under increasing demands [1-4]. 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 programming scheme called the time-triggered message triggered object programming scheme [3, 5, 6]. In the course of developing a RT system engineering methodology based on this TMO programming 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 programming approach are reviewed and then in section 3, a brief overview is taken of the particular programming

<sup>†</sup> 준 회원 : 조선대학교 대학원 컴퓨터공학과

<sup>††</sup> 정 회원 : 조선대학교 컴퓨터공학과 교수

논문접수 : 2002년 8월 21일, 심사완료 : 2003년 6월 2일

scheme. This programming scheme called the time-triggered message triggered object (TMO) programming scheme [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 programming.

## 2. Distributed time triggered simulation framework

Since an RT simulator must exhibit the timing behavior which is the same as or very close to that of the simulation target, the simulator clock must tick at a steady rate. Each tick of the simulator clock is commenced and administered by referencing an RT clock in the simulation execution engine. The ticking rate of the simulator clock in an RT simulator must be chosen such that during any ticking interval, all events which should be simulated during that interval may be transparent to the user of the RT simulator and only the resulting state of the simulator at the end of the ticking interval may be seen by the user. Therefore, the microscopic order in which events are simulated during a ticking interval should be of no concern to the simulator user. This requirement is called the simulator clock atomicity requirement [5, 6]. All computational activities taking place during a ticking interval of the simulator clock maybe viewed as one simulation-step.

In distributed RT simulation, simulator objects are distributed among multiple nodes. Synchronization of the simulation steps of distributed simulator objects in then a key challenge. In other words, a simulation step executed by every member of the distributed simulator object group must be synchronized with the corresponding simulation step executed by any other member. In other word, the simulator clock for one simulator object must commence the  $n$ -th tick neither before the  $(n-1)$  th tick by the clock for another simulator object nor after the  $(n+1)$ -th tick by the clock for another simulator object.

Therefore, every member must perform some activities necessary to stay synchronized with other members. For example, distributed nodes may exchange completion reports at the end of each simulation-step. However, this is not an efficient approach when the number of nodes used is large. The essence of the distributed time-triggered simulation approach is the following :

- ① Every node is equipped with an RT clock and executes each simulation step upon reaching of the RT clock at the predetermined value ; and
- ② Every simulation step is designed to be completed within on ticking interval.

The distributed time triggered simulation approach has major advantages over other distributed simulation approaches, even if we assume that the latter approaches can be adapted somehow to enable RT simulation. This is because synchronization of simulation steps executed by distributed simulator objects under the distributed time triggered simulation scheme does not require message exchanges among the host nodes. The advantages become decisive in heavy load distributed simulation situations.

However, even with the distributed time triggered simulation approach, exchanges of message that represent movements of certain simulation targets form the territory covered by one TMO to the territory covered by another TMO are inevitable. Therefore, the ticking interval must be long enough to cover this kind of message exchanges.

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

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-

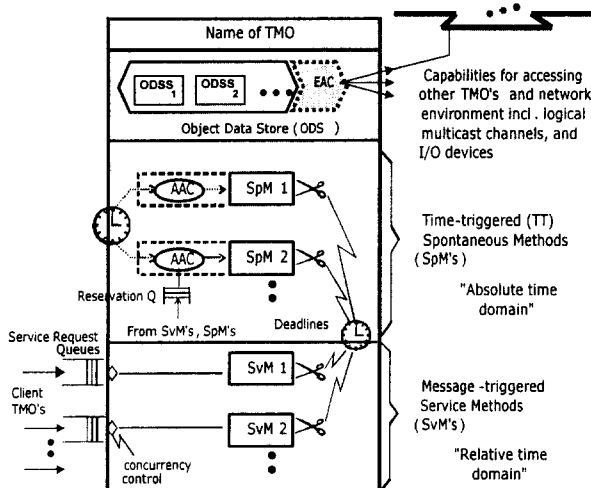
windows for output actions : TMOs are devised to contain only high-level intuitive and yet precise expressions of timing requirements. Priorities are attributes often attached by the OS to low-level program abstractions such as threads and they are not natural expressions of timing requirements. Therefore, 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 (Figure 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.



(Figure 1) Structure of the TMO

① 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.

② 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.

③ 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 or concurrent SvM executions.

④ 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 10 am to 10 : 50 am every 30 min  
start-during (t, t + 5 min) finish-by t + 10 min”

which has the same effect as

“start-during (10 am, 10 : 05 am) finish-by 10 : 10 am”,

“start-during (10 : 30 am, 10 : 35 am) finish-by 10 : 40 am”

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 every 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. Interaction among RT objects and RT message communication

##### 4.1 Non-blocking call

An underlying design 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 is 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 client 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 from 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 from 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 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 from which SvM it will receive returned results, it is implemented to accept results without having to know which SvM the results came from.

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 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 from the called SvM and the case where results are returned from another SvM.

Actually, one can take the view that accepting results returned from the called SvM is a special case of accepting results returned from 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. Many 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 of involved objects. See (Figure 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, then 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 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 from every TT method is also a part of the guarantee.

An output action of a service method may 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
  - ①-a the types of input parameters that the server object can accept and
  - ①-b the maximum request acceptance rate, i.e., the maximum rate at which the server object can receive service requests from 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 from 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 from the arrival of a service request from a client object to the initiation of the cor-

responding service method by the server object ;

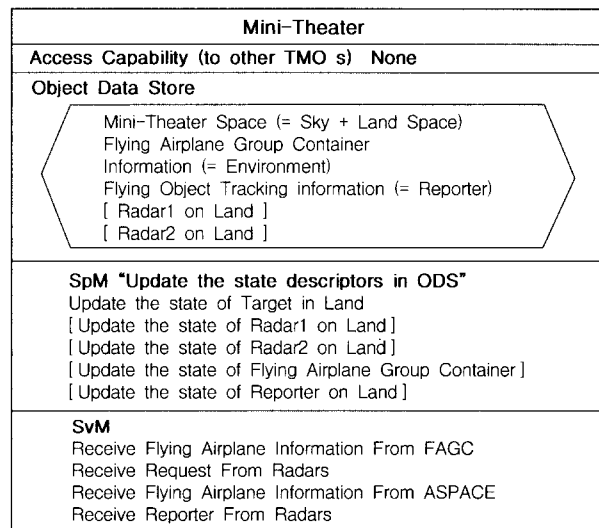
- ② The worst-case execution time for the service method from its initiation to each of its output actions.

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.

### 5. 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.



(Figure2) High-level specification of the Mini Theater TMO

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 :

- 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 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 de-

termined by the control theory logic adopted. In this experimental development, we deployed one control system such as Reporter.

Control Computer System In Reporter	
<b>Access Capability (to other TMOs)</b>	Radar (Accept_spot_check_request)
<b>Object Data Store</b>	Radar data received, Flying airplane tracking information
<b>SpM</b>	<p><b>SpM1</b> Radar Data Processing Step</p> <ul style="list-style-type: none"> <li>• "Process all the radar data received since the last processing cycle, update the flying object tracks"</li> </ul> <p><b>AAC</b> : 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</p> <p><b>InputSpec</b> : Radar data received in the object data store  <b>OutputSpec</b> : &lt;deadline : xxx msec&gt; Reflect changes onto the object data store,                      i.e., Radar data received, Flying airplane tracking info.                      &lt;deadline : xxy msec&gt; Send spot-check radar requests to Radar if</p>
<b>SvM</b>	<p><b>SvM1</b> Receive_from_Radar_on_Land (pos_list)</p> <p>&lt; <b>Accept-with-Delay_Bound-of</b>                      ACCEPTANCE_DEADLINE <b>under</b> MAX_REQUEST_RATE                      finish-within EXECUTION_TIME_LIMIT &gt;</p> <ul style="list-style-type: none"> <li>• "Receive from Radar_on_Land the information on all recent detections."</li> </ul> <p><b>InitiationCond</b> : Other SvM1 invocations are not in place.  <b>InputSpec</b> : pos_list = array of (return_type                      (= scan_search/spot_check), position, time, predicted_time)  <b>OutputSpec</b> : &lt;deadline : yyy msec&gt; Deposit the radar data received in the object data store  <b>SvM2</b> Accept Advice from &lt; <b>Accept-via</b> - .&gt;</p>

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

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.

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.

## 6. 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*, pp.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.
- [8] K. H. Kim and Liu, J., "Deadline Handling in Real-time Distributed Objects," *Proc. ISORC 2000, Newport Beach, CA*, pp.7-15, March, 2000.
- [9] K. H. Kim., "APIs Enabling High-Level Real-Time Distributed Object Programming," *to appear in IEEE Computer*, June, 2000.
- [10] G. J. K, S. D. Na and C. S. Bae, "Time Service Guarantee in Real-Time Distributed Object Oriented Programming of TMO," *Proc. ICIM'01*, pp.215-219, Nov., 2001.



### 송 선 희

e-mail : ssh6061@netian.com  
 1988년 전남대학교 공학학사  
 2002년 호남대학교 공학석사  
 현 재 조선대학교 컴퓨터공학과 박사과정  
 관심분야 : TMO, 실시간통신, 데이터통신망, 이동통신, 멀티미디어 등



### 나 상 동

e-mail : sdna@mail.chosun.ac.kr  
 1968년 조선대학교 공학학사  
 1980년 건국대학교 공학석사  
 1995년 원광대학교 공학박사  
 1998년 조선대학교 전자계산소 소장 역임  
 1995년~1996년, 2001년~2002년 Dept. of Electrical & Computer Eng. Univ. of California Irvine 연구교수

1973년~현재 조선대학교 전자·정보공과대학 컴퓨터공학과 교수  
 관심분야 : TMO, 실시간통신, 디지털통신, 데이터 통신망 및 신호처리 등임