특별기고 <해외편>

Time-triggered Message-triggered Object Programming Scheme and Its Support Middleware

K.H. (Kane)Kim* M.H. Kim** K.W. Rim***

목 차

1. Introduction
2. Motivations for using the OO RT distributed programming approach
3. TMO scheme for high-level OO RT distributed programming
4. Middleware supporting TMO-structured programs
5. Recent TMO related research works
6. Summary

This article was edited by Professors M.H. Kim and K.W. Rim, adapting some materials from the paper[1] and some materials recently provided by Professor K.H. (Kane) Kim.The purpose of this article is to introduce the time-triggered message-triggered object (TMO) programming scheme, which is a powerful and easy-to-use scheme suitable fordeveloping a wide range of real-time applications from complex real-time systems to small embedded real-time systems. The editors (M.H. Kim and K.W. Rim) hope that this article could provide the readers some valuable insights into the potential impacts that this modern real-time distributed programming technology might have on construction of ubiquitous computing societies in the next decade and beyond. Section 5 is given by the editors to help readers get some feel about open research directions on TMO.

* Professor of the University of California, Irvine, USA.
** Professor of the Konkuk University, Korea.
*** Professor of the Sunmoon University, Korea.

1. Introduction

Among several cutting-edge technology movements initiated in 1990's in software engineering is the high-precision real-time(RT) object-oriented(OO) programming movement [2-11]. In our view, the most important goal of that movement has been to instigate a quantum productivity jump in software engineering for RT computing application systems. Particularly targeted application domains have been those challenging large-scale distributed / parallel computing applications in fields such as factory automation, telecommunication, defense, intelligent transportation, emergency management, etc.

As construction of ubiquitous computing societies has become a subject of serious national interests in many advanced countries and along with it demands on RT distributed computing (DC) applications are growing fast, it seems timely to reflect a bit on the state of the art and future directions. The RT OO DC programming movement is still in its youthful stage and its

impact has just started surfacing up. However, its great potential is now much more clearly and widely recognized than it was in 1990's.

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 scheme which the first author and his collaborators have been establishing. This high-level RT DC object programming scheme is called the Time-triggered Message triggered Object (TMO) programming scheme.

The desirable features of middleware providing execution support for OO RT distributed programs are discussed in Section 4. In section 5, some of recent TMO-related research works are described and, finally, section 6 summarizes this article.

2. Motivations for using the OO RT distributed programming approach

2.1 Needs of modem RT distributed programming technologies in constructing ubiquitous computing societies

Starting in mid-1990's, the field of RT computing applications has been showing a rapid growth pattern. Computer systems in those application domains are generally responsible for RT controlof physical devices, RT storage and search for information, and RT communication and display of information. In addition, they are often tasked to perform RT simulation of their application environments. The field of computer-embedded communication-equipped system engineering has been growing particularly fast in recent years.

Now it is a fashion and a subject of national pride in advanced countries to build ubiquitous computing societies. The parts of the ubiquitous computing societies which require new-generation RT distributed programming technologies for their construction are bound to increase rapidly in coming years. Visionaries have been drawing pictures of societies where old weak people living without companions are continuously monitored by nearby as well as remote machines for their health-related behavior and receive timely rescue serviceswhenever they fall into such conditions. In such advanced societies, substantial parts of the actions taken currently by human drivers of cars, including safe-critical actions, may also be delegated to intelligent machines. With conventional RT programming styles and associated methodologies that have been practiced by programmers using assembly languages, C, or other similar low-level languages, it is very difficult to construct safe and dependable ubiquitous computing societies of the types sketched above.

As a result, industry has felt an acute need for RT distributed programming and software engineering methods which are at least multiple times more effective than currently widely practiced programming techniques. Continuous use of old low-level programming styles is not economically viable for dealing with increasing demands for new RT application systems. The new-generation RT distributed software engineering method must be based on a "general

high-level programming style" which can be accommodated with minimal efforts by current-generation business application programmers (using C++ and Java) rather than on a style that has been practiced by assembly language or low-level language programmers.

Designers must be required to specify both the interactions among DC program components and the timing requirements of various actions in natural intuitively appealing forms only. The fact that distributed objects represent a higher-level structure for DC programs than distributed processes do have been widely recognized by the industry in the past 12 years, e.g., technology movements such as CORBA[11], DCOM[12], RMI[13], and .NET[14]. Naturally, quite a few researchers started at various points in the past 12 years searching for extensions of distributed objects that allow unambiguous specification of timing requirements imposed on various computations units[2-11, 17, 18]. However, up to now the industry practice in designing RT computer systems has been predominantly to structure software as a set of concurrent processes and assign fixed priorities to the processes.

The essence of RT computing is to effect important output actions within precisely specified time-windows. In RT DC systems, if an output action of a node becomes ready only after manipulating some data coming from another node, then the production of the data by the latter node and its communication to the former node must all occur in time for enabling the output action to be within the specified time-window.

Therefore, the top-level output deadlines shoulddrive the selection of deadlines for various computational milestones such as completionof a certain function or object method, a message generation by a process, a message pickup by a process, etc[15]. The latter deadlines may be viewed as intermediate deadlines. It is thus prudent and safe to use execution resources in the direction toward minimizing the probability of missing any of these intermediate deadlines and of course, top-level output deadlines. Given this accepted as the main goal, fixed-priority process structuring is a highly indirect and ineffective way for approaching the goal. When we divide various computational responsibilities for accomplishing the RT application into a group of processes, each process will generally be subject to multiple intermediate deadlines, each imposed on a different milestone within the process. Translating such multiple deadlines into an effective single fixed-priority number attached to the entire process is impractical in most cases !

To put it differently, modern RT DC applications are much more complicated than what can be adequately represented by fixed-priority processes. First of all, typical applications involve multi-step fusion of data from distributed sensors. Fusion of new sensor data with the contents of a database, e.g., historical records on sensor data, is also of frequent necessity. Secondly, more often than not, it is necessary to facilitate sharing of data between periodic, or more generally, time-triggered(TT) data acquisition operations and event-triggered reactive operations.

It is thus clear from the above discussions that

it is natural for RT application programmers to think about start-time-windows and completion deadlines of RT computation-segments rather than priority numbers. The timing requirements in most modern RT DC applications cannot be expressed in understandable forms as functions of priorities[15, 16] !

Therefore, designers should not be forced to deal directly with notions such as priorities for expressing application timing requirements. Some people may suggest that designers need to assign priority numbers to computation-segments because if computing resource failures occur and the resulting resource shortage dictates sacrificing the abilities to meet all timing requirements of all computation-segments, less important computation-segments should be sacrificed [19]. If such a decision is represented by assignment of low priorities to the victimized computation-segments, then the priority is really a number representing criticality or importance rather than a timing requirement. Expressing criticality is an issue orthogonal to that of expressing timing requirements. Expressing criticality can be combined with more appropriate approaches for expressing timing requirements such as specifying in terms of start-time-windows and completion deadlines of RT computation-segments.

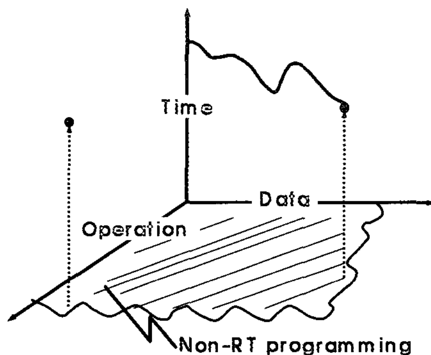
Priorities are usually associated with low-level computation units such as processes and threads and may be used inside the execution engine than directly manipulated by human users. If a DC application is designed in the form of a network of DC objects and a certain object method needs to be started within a certain time-window and completed by a certain deadline, the start-time-window and the completion deadline should be expressed in a natural form as parts of the DC object design. It is a very difficult job for the application designer to assign priority numbersto the objects and methods, hoping that they will reflect the desired start-time-windows and completion deadlines.

2.2 Should RT programming remain an esoteric branch of computer science and engineering ?

It is fair to say that up to now, RT programming hasbeen treated as an esoteric branch of computer science and engineering. Very few universities have courses on RT programming and even those few existing courses are almost entirely graduate courses.

The main reason is that RT programming has been practiced largely as an ad hoc art in a form looking quite alien to the vast number of business and scientific application programmers. On the other hand, there is no reason why future RT computing cannot be realized in the form of a generalization of the non-RT computing, rather than the other way around. (Figure 1) depicts this. If the main-stream (traditional) programming science is viewed as a study of the two-dimensional space, (data operation), then a proper form of RT programming should be practiced as work within the three-dimensional space, (data operation time). Of course, the less the programmer is burdened with the work on the time dimension, the better off. We just need a powerful programming scheme capable of dealing with all practically useful RT and non-RT

computing requirements in uniform manners. Under such a properly established RT programming methodology, every practically useful non-RT program must be realizable by simply filling the time constraint specification part with the default value "unconstrained".



(Figure 1) RT programming as a generalization of non-RT programming

2.3 Reliability of RT distributed programs

RT programs have been notoriously difficult to analyze. It is well known that testing alone does not ensure sufficiently high reliability of RT programs. Given rapidly increasing demands for RT application systems and the fact that complexities of RT DC programs are far greater than those of single-node programs, thepractice of relying solely on testing for reliability assurance is becoming less and less tolerable.

Whether we like it or not, there are certain hard deadlines in human societies and violation of these deadlines have severe consequences. For example, suppose cars are to be driven by automated drivers (robots). If such cars are heading toward a collision course, then the collision can be avoided only if at least one driver detects the danger and takes an avoidance action within a certain hard deadline. Applications subject to hard deadlines are called hard-real-time (HRT) applications. Again, the timely service capabilities of such HRT application systems cannot be sufficiently assured by the testing approach alone.

A new-generation RT distributed software engineering method must thus allow some system engineers dealing with safety-critical applications to confidently produce certifiable RT distributed computing systems. The general public which has witnessed conspicuous improvements in the reliability of the desk-top computer systems in 1990's will demand in this new century a different level of reliability for the systems in safety-critical applications. They will demand sufficiently trustable certifications of the designs and implementations.

Design-time guaranteeing of service and response times of computing components / systems is considered a major technological requirement that must be fulfilled before such certification becomes a common practice[20]. Guaranteeing of service times cannot be done if application software is structured in undisciplined manners. That is, easily analyzable structuring of application software must be pursued to the maximum extent possible. Research in recent years has made it clear that high-level structuring in the form of RT DC objects has significant advantages in this regard in comparison to somewhat lower-level distributed process structuring.

3. TMO scheme for high-level OO RT distributed programming

The TMO scheme is a general-style RT distributed computing (DC) extension of the pervasive OO design / programming approach [6, 7, 15, 21, 22]. It has been established to facilitate RT DC software engineering in a form which software engineers experienced in the vast non-RT software field can adapt to with small efforts. Calling the TMO scheme a high-level distributed programming scheme is justified by the following characteristics of the scheme:

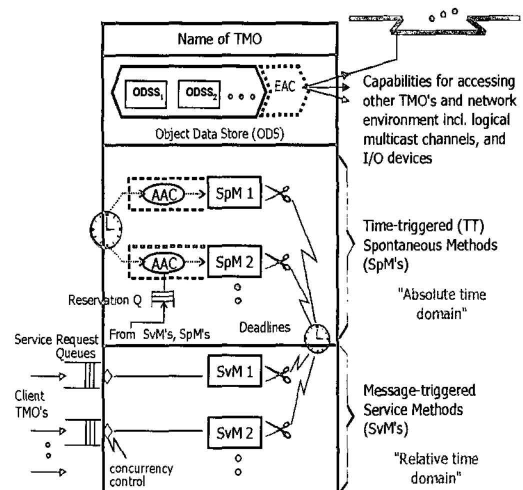
(1) No manipulation of processes and threads: Concurrency is specified in an abstract form at the level of object methods. Since processes and threads are transparent to TMO programmers, the priorities assigned to them by the execution engine, if any, are not visible, either. Yet, TMO offers a powerful structure which is capable of representing all conceivable practical RT and non-RT applications in easy-to-analyze forms.

(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, e.g., sockets,and any direct manipulation of physical channels and physical node addresses / names.

(3) Timing requirements need to be specified only in the most natural form of a time-window for execution of every significant output action, a time-window for every time-triggered method execution, and a completion deadline for every client-requested method execution. This high-level expression matches the most closely with the designer's intuitive understanding of the application's timing requirements.

At the same time the TMO scheme wasdesigned to enable a great reduction of the designer's efforts in guaranteeing timely service capabilities of DCapplication systems. To our knowledge the TMO scheme is one of the very few practical RT object programming schemes that have 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 [20, 23, 24].

3.1 TMO structure and design paradigms



(Figure 2) The Basic structure of TMO(adapted from[7])

TMO is a natural and syntactically minor but semantically powerful extension of the conventional object(s)[6, 7, 15, 21, 22]. As depicted in (Figure 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.

Significant extensions realized by the TMO scheme are summarized below.

3.1.1 Globally referenced time base

All time references in a TMO are references to global timein that their meaning and correctness are unaffected by the location of the TMO[25]. If GPS receivers are incorporated into the TMO execution engine, then a globally referenced time base (or global time base, for short) of a microsecond-level precision can be established easily. Within a local area network a master-slave scheme, which involves time announcements by the master node and exploitation of the knowledge on the message delay between the master node and the slave node, can be used to establish a global time base of the precision ranging from a few microseconds to hundreds of microseconds depending the network interface components and the clock synchronization software used. Over a wide-area network where GPS receivers are not universally available, a master-slave scheme can be used to establish a global time base of a sub-millisecond level precision. A TMO instantiation instruction may contain a parameter which explicitly indicates the required precision of the global time base to be established by the TMO execution engine.

3.1.2 Distributed computing component

TMO's 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 or different nodes, client methods are allowed to make non-blockingtypes of service requests to server methods.

3.1.3 Spontaneous method(SpM) and its clear separation from the service method

The TMO may contain a new type of methods, spontaneous methods (SpM's) (also called time-triggered methods or TT methods), which are clearly separated from 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. Triggering times for SpMs must be fully specified as constants during the design time. Those RT 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".

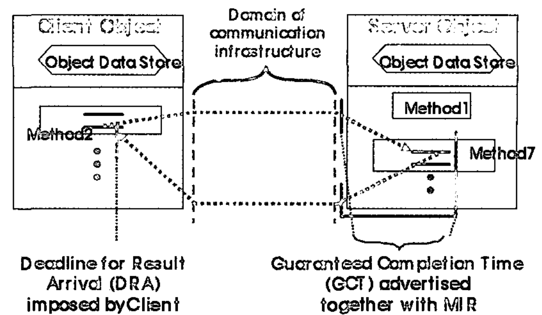
A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an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

A TMO which contains an SpM can also be viewed as an autonomous active DC component.

3.1.4 Basic concurrency constraint (BCC):

This rule prevents potential conflicts between SpM's and SvM's and reduces the designer's efforts in guaranteeing timely service capabilities of TMO's. The full set of data members in a TMO is called an object data store(ODS). An ODS is declared as a list of ODS segments (ODSS's), each of which is thus a subset of the data members in the ODS and is accessed by concurrently running object method executions in the concurrently-reading and exclusive-writing mode. 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 if no SpM that accesses the same ODSS's to be accessed by this SvM has an execution time-window that will overlap with the execution time-window of this SvM. The BCC does not reduce the programming power of TMO in any way.

3.1.5 Guaranteed completion time(GCT) and deadline for result arrival



(Figure 3) Client's deadline vs. Server's GCT with maximum invocation rate (adapted from [21])

Deadlines are handled in the most general form [21, 24]. Basically, for output actions and completion of a method of a TMO, the designer guarantees and advertises execution time-windows boundedby start times and completion times. By advertising these time-window specifications to the designers of potential client objects, the designer of the server TMO guarantees the timely services of the TMO. Before determining the time-window specifications, the server object designer must make sure that with the available object execution engine (hardware +OS) the server object can be implemented such that the output actions are performed within the time-windows. The BCC contributes to major reduction of these burdens imposed on the designer.

On the other hand, the client imposes a deadline for result arrival(DRA) as depicted in (Figure 3). The client object in the middle of executing its method, Method 2, calls for a service, Method 7 service, from the server object. In order to complete its execution of Method 2 within a certain target amount of time, the client must obtain the service result from the server

within a certain deadline. During the design of this client object, the designer searches fora server object with a GCT acceptable to him/her. The designer must also consider the time to be consumed by the communication infrastructure in judging the acceptability of the GCT of a candidate server object.

There are three sources from which a faultmay arise to cause a client's deadline to be violated. They are (s1) the client object's resources which are basically node facility (hardware + OS), (s2) the communication infrastructure, and (s3) the server object's resources which include not only node facility but also the object code. Thus while the server is responsible to finish a service within the guaranteed service time, the client is responsible for checking if the result comes back within the client's deadline or it does not due to a fault. This support for programmers to specify a DRA in association with a remote method call enables "systematic composition" of higher-level services with timeliness assurances out of lower-level services.

Middleware which together with node OS's and hardware make up TMO execution engines, have been developed [26-28]. These and other middleware supporting OO RT programs will be discussed in Section 4.

3.2 TMO structuring in environment modeling and multi-step multi-level design and implementation

The attractive basic design style facilitated by the TMO structuring is to produce a network of TMOs meeting the application requirements in a top-down multi-step fashion[7, 23]. The engineering of an application system can start with a single TMO representation of the entire application environment (including the computer system to be designed) and proceeds through step-by-step expansion of the initial single TMO model toward a final implementation in the form of a network of TMO's executing on engines. This top-down process can also produce a RT simulator of the application environment, again in the form of a TMO network. In fact, the TMO scheme facilitates an attractively simple approach to parallel and distributed RT simulation, called the distributed time-triggered simulation (DTS) [1, 7, 29, 30, 31]. This systematic approach has been shown to be practical via several experiments which dealt with RT DC applications such as a missile defense command-control application [7] and a freeway car traffic control application[29, 31].

4. Middleware supporting TMO-structured programs

We have been enabling TMO programming without creating any new language or compiler. A cost-effective way to support execution of OO RT distributed programs is to realize an execution engine by developing middleware running on well established commercial software / hardware platforms. To support efficient execution of TMO-structured programs, a middleware architecture, named the TMO Support Middleware(TMOSM), has been developed. TMOSM can be easily adapted to a variety of commercial kernel+hardware platforms compliant with industry standards[27, 28]. TMOSM uses

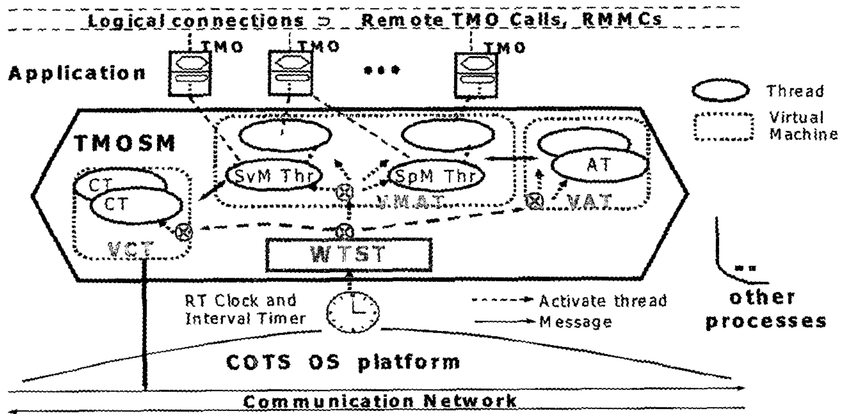
well-established services of commercial OSs, e.g., process and thread support services, short-term scheduling services, and low-level communication protocols, in a manner transparent to the application programmer. The TMOSM architecture was devised to contribute to simplifying the analysis of the execution time behavior of application TMOs running on TMOSM.

TMOSM has been foundto be easily adaptable to most commercial hardware + kernel platforms, e.g., PCs or similar hardware with Windows XP, Windows CE, Linux, etc. A prototype implementation on Windows XP/2000/NT, TMOSM/XP(initially TMOSM/NT), was developed[27]. Our experiences indicate that even this middleware extension of a general-purpose OS (Windows XP) can support application actions with the 10ms-level timing accuracy. TMOES/AnyORB/NT [28] is another prototype implementation realized in the form of a CORBA service that runs on platforms equipped with Windows NT and an ORB (object request broker) and supports CORBA-compliant application TMOs.

A Windows CE based prototype of TMOSM has also been developed and under continuous optimization with the goal of supporting application actions with better-than-10ms-level timing accuracy. It has been used in supporting a few advanced applications, e.g., control software for autonomous ground vehicles, a music ensemble performed by networked musical PCs, each making a unique contribution to a stereo music play, etc. Such application demonstrations are expected to grow at a faster rate from here on. In addition, two Linux-based prototype implementation of TMOSM have been developed. The first one was developed by the second co-author and his collaborators in Korea a few years ago[32] and the second one was developed at the first co-author's location recently. A joint effort by both teams for establishing a unified Linux-based implementation of TMOSM is under way.

TMOSM was devised to support the execution of TMOs with the use of a minimal amount of computing and communication resources to the satisfaction of all the timeliness requirement specifications embedded in TMOs. TMOSM uses well-established services of commercial OS's, e.g., process and thread support services, short-term scheduling services, and low-level communication protocols, in a manner transparent to the application TMO programmer. While devising the TMOSM architecture, an emphasis was on making both the analysis of the worst-case time behavior of the middleware and the analysis of the execution time behavior of application TMOs as easy as possible without incurring any significant performance drawback. As a result, use of mechanisms such as semaphore which leads to frequent blockings of threads inside the middleware was avoided completely and instead, a new extension of the Non-Blocking Writer mechanism invented by Hermann Kopetz [33], called the Non-Blocking Buffer(NBB) mechanism, was used extensively[15, 34].

As depicted in (Figure 4), within TMOSM, the innermost core is a super-micro thread called the



(Figure 4) Virtual machines and threads in TMO

WTST(Watchdog Timer & Scheduler Thread). It is a "super-thread" in that it runs at the highest possible priority level. It is also a "micro-thread" inthat it manages the scheduling / activation of all other threads in TMOSM. Even those threads created by the node OS kernel before TMOSM starts are executed only if WTST allocates some time-slices to them. Therefore, WTST is in control of the processor and memory resources with the cooperation of the node OS kernel.

WTST leases processor and memory resources to three virtual machines(VMs) in a time-sliced and periodic manner. Each VM can be viewed conceptually as being periodically activated to run for a time-slice. Each VM is responsible for a major part of the functions of TMOSM. Each VM maintains a number of application threads. In fact, whenever WTST assigns a time-slice to a VM, the VM in turn passes the time-slice onto one of the application threads belonging to itself. The component in each VM that handles this "time-slice relay" is the application thread scheduler. For example, VM-A has the application-thread-scheduler VM-A-Scheduler. The application thread scheduler is actually executedby WTST. To be more precise, at the beginning of each time-slice, a timer-interrupt results in WTST being awakened. WTST then determines which VM should get this new time-slice. If VM-A is chosen, WTST executes VM-A-Scheduler and as a result, an application thread belonging to VM-A is activated to run for a time-slice as WTST enters into the event-waiting mode.

The set of VMs is fixed at the TMOSM start time. One iteration of the execution of a specified set of VMs is called a TMOSM cycle. For example, one TMOSM cycle may be: VCT VMAT VAT VMAT. The following three VMs handle the core functions:

4.1 VCT (VM for Communication Threads)

The application threads maintained by this VM are those dedicated to handling the sending and receiving of middleware messages. Middleware messages are exchanged through the communication network among the middlewar

instantiations running on different DC nodes to support interaction among RC DC objects, i.e., TMOs. Therefore, these application threads are called communication threads and denoted as CTs in (Figure 4). A communication thread also distributes middleware messages coming through the network to their destination threads, typically belonging to another VM discussed below.

4.2 VMAT (VM for Main Application Threads)

The application threads maintained by this VM are those dedicated to executing methods of TMOs with maximal exploitation of concurrency. Those application threads are called main application threads and denoted as MATs in (Figure 4). Normallyto each execution of a method of an application TMO is dedicated a main application thread. In principle, TMO method executions may proceed concurrently whenever there are no data conflicts among the method executions. Every time-slice not used by the other VMs is normally given to this VM. In every one of our prototype implementations of TMOSM, the application thread scheduler in VMAT uses a kind of a deadline-driven policy for choosing a main application thread to receive the next time-slice [35].

4.3 VAT (VM for Auxiliary Threads)

This VM maintains a pool of threads which are called auxiliary threads and denoted as ATs in (Figure 4). Some auxiliary threads are designed to be devoted to controlling certain peripherals under orders from TMO methods (executed by main application threads). Others wait for orders for executing certain application program-segments and such orders come from main application threads in execution of TMO methods. Use of this VAT has been motivated partly by the consideration that it should be easier to analyze the temporal predictability of the application computations handled by each VM, i.e., those handled by VMAT and those by VAT, than to analyze the temporal predictability of the application computations when there is no VAT and thus VMAT alone handles the combined set of application computations.

Also, WTST provides the services of checking for any deadline violations and if a violation is found, it provides an exception signal to the user.

We believe that structuring of VMs as periodic VMs is a fundamentally sound approach which leads to easier analysis of the worst-cast time behavior of the middleware without incurring any significant performance drawback.

A friendly programming interface wrapping the execution support services of TMOSM has also been developed and named the TMO Support Library (TMOSL)[21-23]. It consists of a number of C++ classes and approximates a programming language directly supporting TMO as a basic building-block. The programming scheme and supporting tools have been used in a broad range of basic research and application prototyping projects in a number of research organizations and also used in an undergraduate course on RT DC programming at UCI for about three years [http://dream.eng.uci.edu/ece147/serious.htm]. TMO facilitates a highly abstract programming style without compromising the

degree of control over timing precisions of important actions.

4.4 Other middleware supporting OO RT distributed programs

OMG has been developing the RT CORBA specifications in the past few years. They decided to proceed in two steps. First, they produced the version based on static priority assignment[11]. An ORB meeting this specification has also been produced[9, 36]. This step has been justified on the grounds that many current-generation RT programmers who deal with safety-critical applications and use low-level programming languages may have easier time adapting to such styles of programming distributed objects. However, we feel that static priority assignment does not match well with RT DC object programming[15, 16, 19]. It defeats the goal of adopting the high-level OO style into RT DC programming. Recognizing the limitation of the first version, OMG started the second step of developing a specification based on dynamic scheduling a few years ago[37]. However, this effort appears to be progressing quite slowly.

In facilitating the TMO programming without involving a new language translator, it was necessary to rely on the programmer for groupingall parameters into a single structured variable and program the client object to pass a pointer for the variable along with the information on the size of the memory area of the variable onto the execution engine. This restriction is removed in the CORBA which uses an IDL (interface definition language) translator[11]. The programmer of a CORBA object class produces an IDL specification, which contains the method names and method parameters, in addition to the class. An IDL translator then takes the IDL specification as an input and produces two program-modules, one called the stub for use by the client objects and the other called the skeleton for use by the server object. The stub-skeleton pair takes care of parameter transfer across the network and may perform multiple message exchanges to handle a large set of parameters.

As mentioned earlier a CORBA service named TMO execution support (TMOES) that supports CORBA-compliant application TMOs has been defined and a prototype implementation, TMOSM /AnyORB/NT, that runs on platforms equipped with Windows NT and a basic ORB has been obtained[28]. Of course, the API wrapping this TMOES involves the use of IDL and no restriction on parameter structuring is imposed.

Efforts for establishing the specification of distributed RT Java and the support middleware have been under way for several years [8-10] but they appear to be moving slowly. It is not clear exactly when middleware supporting RT DC objects based on the .NET architecturewill appear but we are certain that we will see some in the next few years.This is again because the pressure in industry to find and use new-generation RT DC programming and software engineering methods has already grown to a very serious level.

5. Recent TMO related research works

This section listssome of the recently performed

and on-going research works related to the TMO programming scheme to the editors' knowledge.

(1) TMOSM on Windows CE has been developed by the first author and his collaborators and its optimization efforts are continuing. This toolkit will be made available to the public in early 2005.

(2) The initial framework of amiddleware architecture for supporting reliable fault-tolerant execution of TMO-structured DC applications, was established several years ago and it has undergone multiple enhancement steps since then. This middleware has been named ROAFTS (Real-time Object-oriented Adaptive Fault Tolerance Support)[38-40]. Its prototype implementation is under way.

(3) An attractively simple approach to parallel and distributed RT simulation, called the distributed time-triggered simulation(DTS) [1, 7, 30, 31] scheme, was formulated by the first co-author ten years ago. It can be practiced easily by use of the TMO programming tools. Efforts for expanding the scientific foundation for DTS are under way.

(4) The TMO scheme has been proven effective in developing RT local-area DC application systems. Efforts for extending the TMO scheme into a technology for efficient realization of RT wide-area DC began recently, especially with a plan for a demo in a tightly managed optical Grid environment [19].

(5) Linux-based TMO related research has been conducted by the second co-author and his collaborators [36].

(6) A sensor network software architecture based on the TMO structuring principle has been formulated and a support kernel developed [41].

Besides the activities mentioned above, many active research works related to the TMO scheme are under way in research institutions in various parts of the world. For more information, refer "http://dream. eng.uci.edu/TMO/TMO.htm" or contact the Software Research Center of the Konkuk University in Korea.

6. Summary

OO RT programming is a technology expected to flourish in this quarter of the 21st century. Currently, its youthfulness is indicated by the insufficient availability of the support middleware and the associated API, let alone language compilers. The middleware providing fault-tolerant execution support is in its infancy. The advances in OO RT distributed programming will also enable large-scale RT simulations. The research community dealing with this technology area is expected to grow continuously for foreseeable future and consequent accelerations of the technology advances will in turn accelerate the development of many new types of sophisticated RT DC applications as well as realization of advanced types of ubiquitous computing societies.

References

[1] Kim, K.H., "Object-Oriented Real-Time Distributed Programming and Support Middleware", Proc. ICPADS 2000 (IEEE CS 7th Int'l Conf. on Parallel and Distributed

Systems), Iwate, Japan, July 2000, pp.10-20 (Keynote paper).

[2] Kopetz, H. and Kim, K.H., "Temporal Uncertainties in Interactions among Real-Time Objects", Proc. IEEE CS 9th Symp. on Reliable Distributed Systems, Oct. 1990, pp.165-174.

[3] Attoui, A. and Schneider, M., "An Object Oriented Model for Parallel and Reactive Systems", Proc. IEEE CS 12th Real-Time Systems Symp., 1991, pp. 84-93.

[4] Takashio, K., and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", Proc. OOPSLA, 1992, pp. 276-294.

[5] Ishikawa, Y., Tokuda, H., and Mercer, C. W., "An Object-Oriented Real-Time Programming Language ", IEEE Computer, October 1992, pp. 66-73.

[6] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", Proc. WORDS '94 (IEEE CS '94 Work. on Object-Oriented Real-Time Dependable Systems), Oct. 1994, Dana Point, pp.36-45.

[7] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", IEEE Computer, Vol. 30, No.8, August 1997, pp. 62-70.

[8] J Consortium, "Real-Time Core Extensions for the Java Platform", Specification No. T1-00-01, Rev. 1.0.10, available from www.j-consortium.org, Feb. 3, 2000.

[9] 'A special issue of Computer (a magazine of IEEE Computer Society) on Object-oriented Real-time distributed Computing', June 2000.

[10] Real Time Specification for Java Experts Group, "Real-time Specification for Java, Version 0.9.2", available from www.rtj.org/public, March 29, 2000.

[11] Object Management Group, "Chapter 24. Real-time CORBA", in CORBA Specification, Version 2.6.1, http://www.omg.org/technology / documents/formal/ corba_2.htm, May, 2002.

[12] Sessions, R., 'COM & DCOM: Microsoft's Vision for Distributed Objects', John Wiley & Sons, Inc., New York, Oct. 1997.

[13] Sun Microsystems, "Java Remote Method Invocation Specification", Revision 1.50, JDK 1.2, Oct. 1998, available from web2.java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmi-title.doc.html .

[14] Platt, David S., 'Introducing Microsoft .NET', ISBN 0-7356-1918-2, 2003, Microsoft Press.

[15] Kim, K.H., "Basic Program Structures for Avoiding Priority Inversions", Proc. ISORC 2003 (IEEE CS 6th Int'l Symp. on Object-oriented Real-time distributed Computing), Hakodate, Japan, May 2003, pp. 26-34.

[16] Kim, K.H., "Questionable Relevancy of Fixed Priority Assignment in Distributed Object Design", Proc. ISORC '99, May 1999, pp. 283-285 (Position paper).

[17] Series of ISORC (IEEE CS Int'l Symp. on Object-oriented Real-time distributed Computing) proceedings.

[18] Series of WORDS (IEEE CS Workshop on Object-oriented Real-time Dependable Systems) proceedings.

[19] Kim, K.H., "Wide-Area Real-Time Computing in a Tightly Managed Optical

Grid An Optiputer Vision", Proc. AINA 2004 (IEEE 18th Int'l Conf. on Advanced Information Networking and Applications), Fukuoka, Japan, March 2004 (Keynote paper).

[20] Kim, K.H., "QoS Certification of Real-Time Distributed Computing Systems: Issues and Promising Approaches", IEICE Trans. on Information & Systems, Vol. E86-D, No. 10, Oct. 2003, pp.2077-2086. (An expanded version of the keynote at IEEE HASE 2002)

[21] Kim, K.H., "APIs Enabling High-Level Real-Time Distributed Object Programming", IEEE Computer, June 2000, pp.72-80.

[22] Kim, K.H., "Commanding and Reactive Control of Peripherals in the TMO Programming Scheme", Proc. ISORC 2002 (5th IEEE CS Int'l Symp. on OO Real-time distributed Computing), Crystal City, VA, April 2002, pp.448-456.

[23] Kim, K.H., "Real-Time Object-Oriented Distributed Software Engineering and the TMO Scheme", Int'l Jour. of Software Engr'g & Knowledge Engr'g, Vol. 9, No.2, April 1999, pp.251-276.

[24] Kim, K.H., Liu, J., and Kim, M.H., "Deadline Handling in Real-Time Distributed Objects:Issues and Basic Approaches", Computer System Science & Engineering, Vol. 16, No. 2, March 2001, pp. 109-117.

[25] Kopetz, H., 'Real-Time Systems: Design Principles for Distributed Embedded Applications', Kluwer Academic Publishers, ISBN: 0-7923-9894-7, Boston, 1997.

[26] Kim, K.H. et al., "The DREAM Library Support for PCD and RTO.k programming in C++", Proc. WORDS '96, Laguna Beach, Feb. 96, pp. 59-68.

[27] Kim, K.H. Ishida, Masaki, Liu, Juqiang, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", Proc. ISORC '99 (2nd IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing), St. Malo, France, May, 1999, pp.54-63.

[28] Kim, K.H., Liu, J.Q., Miyazaki, H., and Shokri, E.H., "CORBA Service Middleware Enabling High-Level High-Precision Real-Time Distributed Object Programming", Computer System Science & Engineering, Vol 17, No 2, March 2002, pp.77-84.

[29] Kim, K.H., Liu, J., and Ishida, M., "Distributed Object-Oriented Real-Time Simulation of Ground Transportation Networks with the TMO Structuring Scheme", Proc. COMPSAC '99 (IEEE CS Computer Software & Applications Conf.), Phoenix, AZ, Oct. 1999, pp.130-138.

[30] Kim, K.H., "The Distributed Time-Triggered Simulation Scheme : Core Principles and Supporting Execution Engine", Real-Time Systems - The International Journal of Time-Critical Computing Systems, Vol. 26, 2004, pp.9-28.

[31] Kim, K.H., and Paul, R., "The Distributed Time-Triggered Simulation Scheme Facilitated by TMO Programming", Proc. ISORC 2001 (4th IEEE CS Int'l Symp. on OO Real-time distributed Computing),

Magdeburg, Germany, May 2001, pp. 41-50.

[32] Konkuk University Software Research Center (directed by M.H. Kim), 'Linux-based Highly Reliable Distributed Real-time Operating Platform and Its Application Development Technology', a research project sponsored by Korea MIC under the university ITRC support program (2000-2007).

[33] Kopetz, H., and Reisinger, J., "NBW: A Non-Blocking Write Protocol for Task Communication in Real-Time Systems", Proc. IEEE CS 1993 Real-Time Systems Symp., Dec. 1993, pp.131-137.

[34] Kim, K.H., "A Non-Blocking Buffer Mechanism for Real-Time Event Message Communication", to appear in Real-Time Systems - The International Journal of Time-Critical Computing Systems.

[35] Kim, K.H., and Liu, J.Q., "Going Beyond Deadline-Driven Low-level Scheduling in Distributed Real-Time Computing Systems", in B. Kleinjohann et al. eds., 'Design and Analysis of Distributed Embedded Systems' (Proc. IFIP 17th World Comp. Congress, TC10 Stream, Montreal, Aug. '02), Kluwer, pp.205-215.

[36] Schmidt, D.C., Levine, D.L., and Mungee, S., "The Design and Performance of Real-Time Object Request Brokers," Computer Communications, vol.21, Apr. 1998, pp.294-324.

[37] Object Management Group, "Dynamic Scheduling Request For Proposal", OMG Document: Orbos/99-03-32, March 26, 1999.

[38] m, K.H., and Subbaraman, C., "Fault-Tolerant Real-Time Objects", Communications of the ACM, January 1997, pp. 75-82.

[39] Kim, K.H., "ROAFTS: A Middleware Architecture for Real-time Object-oriented Adaptive Fault Tolerance Support", Proc. HASE '98 (IEEE CS 1998 High-Assurance Systems Engineering Symp.), Washington, D.C., Nov. 1998, pp.50-57.

[40] Kim, K.H., "Middleware of Real-Time Object Based Fault-Tolerant Distributed Computing Systems: Issues and Some Approaches", Proc. IEEE 2001 Pacific Rim Int'l Symp. on Dependable Computing (PRDC 2001), Dec. '01, Seoul, pp. 3-8 (Keynote paper).

[41] Kim, K.H., Im, C.S., M.C. Kim, Y.Q. Li, S.M. Yoo, and L.C. Zheng, "A Software Architecture and Supporting Kernel for Largely Synchronously Operating Sensor Networks", in B. Kleinjohann et al. eds., 'Design Methods and Applications for Distributed Embedded Systems' (Proc. IFIP 18th World Computer Congress, TC10 Conf. on Distributed & Parallel Embedded Systems (DIPES 2004), Toulouse, France, Aug 2004), Kluwer, pp.133-144.