The TMO Scheme for Wide-Area Distributed Real-Time Computing and Distributed Time-Triggered Simulation

K. H. (Kane) Kim and Stephen F. Jenks

University of California, Irvine
Dept. of EECS
Irvine, CA, 92697 U.S.A.
{khkim,sjenks}@uci.edu

Abstract

Some parts of our recent efforts on establishing a technical foundation for wide-area distributed real-time computing (DRC) and distributed time-triggered simulation (DTS) are briefly reviewed. The basic building-block of our technology framework is the Time-triggered Message-triggered Object (TMO) specification and programming scheme. The TMO scheme for local-area DRC has been established in a sound form and its practicality and attractiveness have been extensively demonstrated. However, its extension to fit into wide-area-network based DRC is in an early stage. The distributed time-triggered simulation (DTS) scheme is a new type of an approach to real-time simulation based on parallel / distributed computing. The TMO scheme facilitates DTS in efficient forms. Recent developments in TMO-structured wide-area DRC and DTS and the supporting tools are briefly reviewed.

1. Introduction

While local area distributed real-time computing (DRC) is a steadily advancing technology field with many immature aspects in its core at this time, wide-area DRC is in its infancy. Efforts to create wide area network (WAN) environments in which bounds on communication delay jitter are significantly smaller than those in most segments of the current Internet, started appearing only in recent years. The OptIPuter research sponsored by the National Science Foundation (NSF) is a good example [13].

In recent years, we have attempted to extend the DRC technology established for use in local area network (LAN) environments to fit into the WAN environments. The basic building-block of our technology framework is the *Time-triggered Message-triggered Object* (TMO) specification and programming scheme [4][5][6]. The TMO scheme includes establishment and use of a *global time base* which provides consistent real-time information available in all distributed computing nodes [12]. The TMO scheme facilitates easy exploitation of the principle of *global-time-based coordination of distributed actions* (TCoDA) which has been promoted the most effectively for more than 20 years by Kopetz [12].

The TMO scheme for local-area DRC has been established in a sound form and its practicality and attractiveness have been extensively demonstrated. However, its extension to fit into wide-area-network based DRC is in an early stage. In this paper, we present a brief review of the progresses made recently in extending the TMO scheme for use in WAN environments with the support of NSF.

A brief review of the progresses made in our research on advancing the technical foundation for real-time simulation is also given here. Major improvements in the validation technology for embedded systems and other types of DRC systems are under increasing demands from industry. Not only description but also simulation of non-computer parts and application environments of DRC systems is needed in validating many DRC system software designs and implementations. Here the desired types of simulators are *real-time simulators* which exhibit the timing behavior that are the same as or sufficiently close to the timing behavior of the simulation targets. Such simulators can enable highly cost-effective testing of the DRC software and 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-real-time simulators of environments.

As the complexities of RT simulators grow, the use of distributed and parallel RT simulation approaches become imperative. However, practical distributed RT simulation techniques have not been established in sufficiently reliable forms for use in practicing fields. In recent years, a new direction for RT simulation which is conceptually simple and easy to use but widely applicable, has been formulated. This theme called the *distributed time-triggered simulation* (DTS) *scheme* [4][7] is highly promising in enabling attractively simple practical approaches to parallel and distributed RT simulation. The

1-4244-0910-1/07/$20.00 ©2007 IEEE

DTS scheme is a byproduct of our past research on the TMO specification and programming scheme.

A number of challenging research issues exist in optimal application of DTS. In particular, maximizing the concurrency in RT distributed simulation while maintaining the consistency of distributed RT simulator nodes is a fundamental challenge. Also, to establish DTS as an economic technology, middleware and application programming interfaces (APIs) that support DTS and simulator programming must be developed in sound forms. The middleware support and APIs associated with the TMO scheme have advanced steadily in the past decade. However, further research is needed to exploit multi-processor systems efficiently. Graphic support is also of great importance and is a subject for much further study.

In this paper, we briefly summarize the progresses achieved by us and our collaborators in the research areas mentioned above. Section 2 deals with the field of wide area DRC while Section 3 deals with DTS. Section 4 is a conclusion.

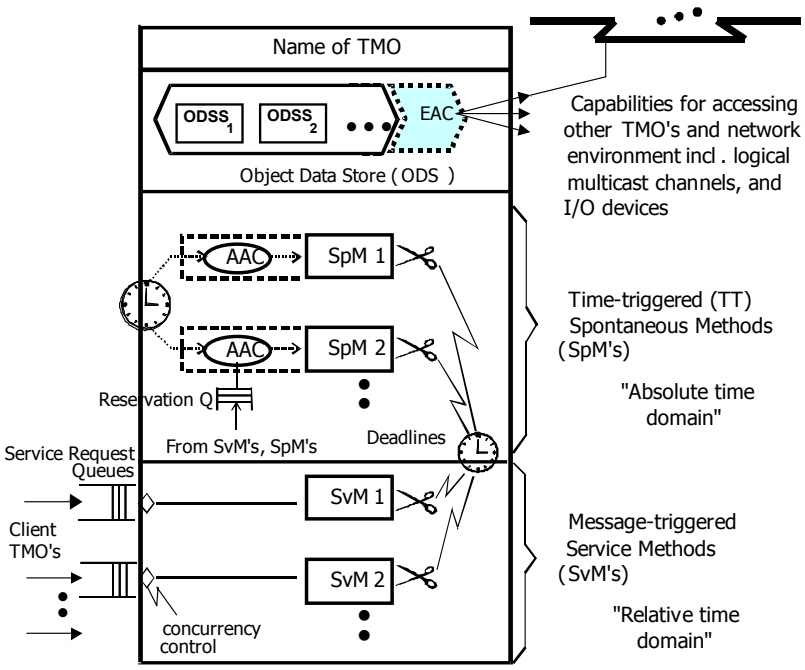

Figure 1. Structure of the TMO (adapted from [4])

2. The TMO Scheme for Wide-Area Distributed Real-Time Computing (DRC)

2.1 The basic building-block of the technology framework: the *Time-triggered Message-triggered Object* (TMO) scheme

The TMO scheme is a general-style DRC extension of the conventional OO design / programming approach. It has been established to facilitate RT distributed software engineering in a form which software engineers in the vast business software field can adapt to with small efforts. It supports a high-level style of DRC programming above the level of abstractions such as processes, threads, and priorities.

TMO structure and design paradigms. A graphical depiction of TMO is given in Figure 1. The key features are as follows.

(TM1) All time references in a TMO are references to *global time* [12] in that their meaning and correctness (e.g., 10am) are unaffected by the location of the TMO.

(TM2) TMO is a distributed computing component. Non-blocking types of remote method calls are supported.

(TM3) TMO has been devised to contain only high-level intuitive and yet precise expressions of timing requirements. *Start-time-windows* and *completion deadlines* for object methods and *time-windows for output actions* are used but no specification in indirect terms (e.g., priority) are required. *Deadlines for result arrivals* can also be specified in the client's calls for service methods.

(TM4) TMO is also an *autonomous active* DC component. Its autonomous action capability stems from one of its unique parts, called the *time-triggered (TT) methods* or the *spontaneous methods* (SpMs), which are clearly separated from the conventional *service methods* (SvMs). The SpM executions are triggered upon reaching of the global time at specific values determined at the design time whereas the SvM executions are triggered by service request messages from clients. For example, the triggering times may be specified as "for t = from 10am to 10:50am every 30min start-during (t, t+5min) finish-by t+10min". By using SpMs, *global time based coordination of distributed actions* (TCoDA), a principle pioneered by our international collaborator Hermann Kopetz [12], can be easily designed and realized.

(TM5) TMOs can use another interaction mode in which messages can be exchanged over logical multicast channels of which access gates are explicitly specified as data members of involved TMOs. The channel facility is called the *Real-time Multicast and Memory-replication Channel* (RMMC) [6].

(TM6) A major execution rule intended to enable reduction of the designer's efforts in guaranteeing timely

service capabilities of TMOs is the *basic concurrency constraint* (BCC) that prevents potential conflicts between SpMs and SvMs. 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* (ODSSs), 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.* Thus an SvM is allowed to execute only if no SpM that accesses the same ODSSs 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.

(TM7) An underlying design philosophy of the TMO scheme is that an RT computer system will always take the form of a network of TMOs, which may be produced in a top-down multi-step fashion, called the TMO Network Development Methodology (TMONDeM) [4][10]. Also, TMO is capable of representing all conceivable practical RT and non-RT applications.

Middleware and APIs that support DTS and simulator programming. We have been enabling TMO programming without creating any new language or compiler. Instead, a middleware architecture called the TMOSM (*TMO Support Middleware*) provides execution support mechanisms and can be easily adapted to a variety of commercial kernel+hardware platforms compliant with industry standards. 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. Prototype implementations running on three major OS kernel platforms, Windows XP, Windows CE, and Linux v2.6, exist (http://dream.eng.uci.edu/TMOdownload/) [1].

A friendly programming interface wrapping the execution support services of TMOSM has also been developed and named the *TMO Support Library* (TMOSL) [6] (http://dream.eng.uci.edu/eecse123/serious.htm). 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 DRC programming at UCI for about four years.

A GUI (graphic user interface) approach to designing an initial skeleton of each TMO and letting a tool generate a code-framework for each TMO, has been formulated and some experiments have been conducted with successful results [10]. The GUI based tool has been named the *Visual Studio for TMO* (ViSTMO). A study on a cost-effective method for realizing high-quality graphic display of the dynamically changing states of TMOs in DTS has also been performed [2].

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. In spite of that, our recent efforts were devoted to much further enhancing the modularity and analyzability of the TMOSM. As a result, a newly enhanced architecture for TMOSM has been developed. 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 [12], called the *Non-Blocking Buffer* (NBB) mechanism [9], was used extensively. Also, TMOSM now consists of the main part which is independent of the OS kernel platform and a small part, called the *Kernel Adaptation Layer* (KAL), which depends on the OS kernel platform chosen.

2.2 Distance-aware TMO (DA-TMO) for use in WAN environments

We recognized that the following issues need to be addressed in order to extend the TMO scheme and establish an efficient approach for designing wide area DRC systems.

With the current TMOSM architecture optimized for use in LAN environments, TMOSM instantiations running on different distributed computing nodes cooperate and interact frequently among themselves. The current TMOSM is expected to show rather poor performance when it is ported to WAN environments without substantial refinement. This is due to the large communication latency inherent in an RT DVC occupying a large geographical region.

As building-blocks of local-area DRC systems, TMOs are treated as all equal neighbors. We decided to extend this notion of a distance-unaware TMO into a newly extended TMO model called the *distance-aware TMO* (*DA-TMO*) in order to establish an effective building-block for wide-area DRC systems. DA-TMO programmers should expect that TMOSM instantiations supporting nearby TMOs will interact with a relatively high frequency whereas TMOSM instantiations supporting TMOs separated by long distances will interact less frequently. They should also expect that a call by a client TMO for a service offered by a remote TMO can involve searches for information not readily available in the local TMOSM instantiation. TMOSM can now be aware of

when an RMMC covers a large geographical area.

TMOSM contains a component called the TMO Network Configuration Manager (TNCM) which is responsible for maintaining the information on the interconnections of distributed computing nodes and the distribution of TMOs on those nodes. Efforts to extend TNCM and other parts of TMOSM to support DA-TMO are underway.

The clock synchronization module of TMOSM has been enhanced to take advantage GPS facilities which serve as a source of global time of micro-second precision. In addition, middleware support components for dynamic creation and destruction of TMOs have been incorporated into TMOSM.

Member sites of a WAN are often machines of PC cluster types. We have thus been developing a version of TMOSM for such a cluster. The clock synchronization module of TMOSM/cluster has been refined to incorporate a broadcast-based clock synchronization algorithm, which provides global time of around 50 microsecond precision to all the nodes in the same cluster. The communication module of TMOSM/cluster has also been enhanced to use multiple plug-in communication facilities, (UDP on Myrinet/Ethernet, MPI on Myrinet/Ethernet).

2.2 High-quality multimedia streaming service

An approach for realizing high-quality tele-audio services over networks by applying the global time based coordination of distributed actions (TCoDA) principle was realized. The goal is to play the audio stream at a remote site with minimal loss of the temporal relationship among the audio data units in spite of the jitters in the transmission delays over networks. The effective programming tool used was the TMO programming tool. Based on this service, a TMO-based audio streaming application over heterogeneous platforms, e.g., Windows XP, Windows CE.NET, and Linux 2.6, was constructed. In LAN-based experiments, the maximum intra-stream jitter was merely 17ms. Further experiments involving both LANs and WANs are under way.

A video streaming service of a similar kind was studied, too, with highly promising results and demonstrations.

2.3 Establishment of wide-area DRC testbeds

A small "robot" car named the *TMO Turtle* which is driven by a remote human user operating a joystick connected to a local PC in the driver's site, has been constructed. Two ITX single-board PCs, one equipped with 802.11 wireless LAN capabilities and the other for future incorporation of additional sensors, have been installed on the car. As depicted in Figure 2, the command messages from the joystick travel via wired networks to a wireless access point near the car and continue through an 802.11 wireless link to reach the ITX PC onboard the car. The remote driver sees the car and its environment via video-streaming from the camera located near the car to the driver's PC station.

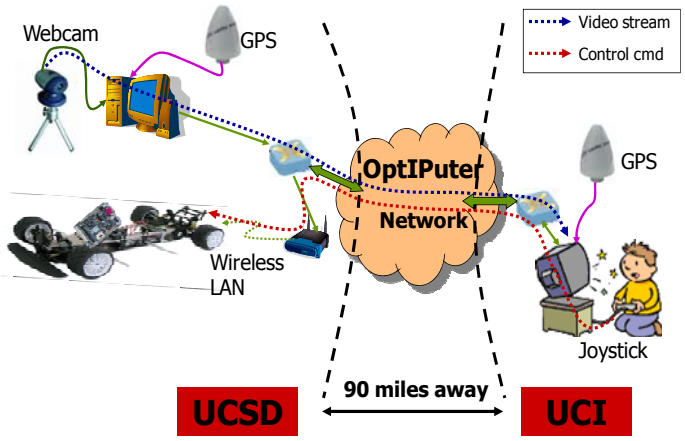

Figure 2. A setup for remote driving of the car, TMO Turtle

The first version of TMO Turtle was used last year in demonstrations of remote driving over the distance of 90 miles between the driver and the car (a video clip available in http://dream.eng. uci.edu/demo/). The car was located at UCSD and the driver was at UCI and a broadband optical network (OptIPuter, http://www.optiputer.net/) connecting the two sites with the 1Gb bandwidth and the low jitter was involved. Application software has been structured as a network of TMOs running on ITX PCs and nearby desktop machines. Measured data showed that application-to-application transmission delays of both the video stream (20 frames of 640 x 480 pixels per second) and the joystick commands (sent every 40 milliseconds) were always less than 60 milliseconds. The car ran at the speed of about 5 - 10 miles per hour. An experiment involving the distance of 6,000 miles will be conducted, starting in January of 2007.

Efforts to increase the autonomous navigation capabilities of TMO Turtle will be stepped up in the near future. We also plan to establish a fleet of three TMO Turtle's in the future.

3. TMO-structured DTS

3.1 Basic requirements in real-time simulation and the DTS scheme

Since a real-time 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. The simulator clock must thus be based on a real-time clock. Making a global time base available to distributed computing nodes economically, whether in local area network or wide area network environments, has become by and large a non-issue due to emergence of GPS, TTP hardware, and other hardware solutions in recent years. Each tick of the simulator clock is commenced and administered by referencing a real-time clock in the *simulation execution engine* (a computer running the simulation program). All computational activities taking place during a ticking interval of the simulator clock may be viewed as one *simulation-step*.

In distributed real-time simulation, *simulator objects (or processes)* are distributed among multiple nodes. *Synchronization of the simulation-steps of distributed simulator objects* is then a key challenge. A simulation-step executed by the distributed nodes as a group must include the activities necessary to keep the executions of the simulation-step by the nodes synchronized. A simulation-step executed by a member of the distributed simulator object group must be synchronized with the corresponding simulation-step executed by any other member. The simulator clock for one simulator object must commence the n-th tick neither before the (n-1) - th tick by the clock driving another simulator object nor after the (n+1) - th tick by the latter clock.

The essence of the *distributed time-triggered simulation* (DTS) approach is the following:

(1) Every node is equipped with a real-time clock and executes each simulation-step upon reaching of the real-time clock at the predetermined value; and

(2) Every simulation-step is designed to be completed within one ticking interval.

The DTS approach has major advantages over other distributed simulation approaches, even if we assume that the latter approaches can be adapted somehow to enable real-time simulation. This is because synchronization of simulation-steps executed by distributed simulator objects under the DTS scheme does not require message exchanges among the host nodes (not counting the message exchanges which may be needed at a certain low frequency for re-synchronizing the real-time clocks of the nodes). The advantages become decisive in heavy-load distributed simulation situations.

However, even with the DTS approach, exchanges of messages that represent movements of certain simulation targets from the territory covered by one simulator node to the territory covered by another node are inevitable. Even such message transmissions can be performed simultaneously (to be more exact, with no serialization or dependency) by all N simulator nodes, provided that the topology of the DRC platforms permit it. For example, in a mesh-connected DRC system, all nodes can simultaneously send data messages to their left neighbors (as well as to right, upper, or lower neighbors). Therefore, the ticking interval must be long enough to cover this kind of message exchanges.

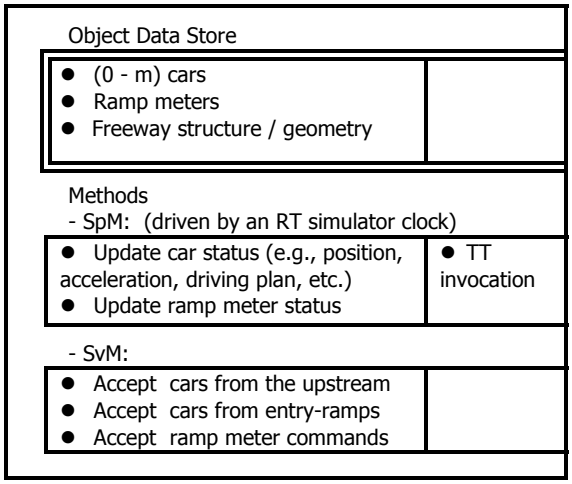

Figure 3. TMO-structured simulation model for freeway segment (Adapted from [Kim04a])

The DTS approach facilitated by the TMO programming scheme uses distributed TMOs of which SpMs execute simulation-steps [4][11]. For example, a freeway-segment can be represented at a high level and simulated by the TMO in Figure 3.

The object data store (ODS) in this TMO contains state representations of the cars, the meters on entry-ramps, and the freeway structure. Each TT method or SpM, when executed, updates a variable-set in the ODS representing the state of some simulation target item (i.e., physical item such as car, ramp meter, etc) to reflect the current state of the target item. Ideally the TT methods should be *activated continuously* and each of their executions be *completed instantly*. However, the limited power of the execution engine dictates the *activation frequency* of any TT method to be a fraction of the ticking rate of the real-time clock in the execution engine. The activation frequency of the TT method may be viewed as *the ticking rate of the target item simulator clock*. Each execution of a TT method must be completed within one ticking interval of the target item simulator clock. Therefore, TT methods are the mechanisms for approximately simulating continuous state changes that occur naturally in the target items in the environment.

A fundamental obstacle in parallel / distributed execution of real-time simulation actions is the *update-dependency*. When a simulation target item covered by one simulator node is update-dependent on another simulation target item covered by another simulator node, update activities of the two nodes must be serialized. In the case of Figure 3, it is a usual practice to sort cars and update them in the sorted order. If the order is to update front cars first and rear cars later, then whenever a value

for the new state of a car is calculated, a check is made whether it is *in conflict* with the already calculated new states of the cars in the front. If a conflict is detected (i.e., if the value is such that it leads the simulator to inconsistency), the value is discarded and a try is made to produce a new value until a conflict-free value is produced. Two simulator objects, e.g., car simulators B and D, in such update-dependent relationship, cannot be updated independently. The update-dependency can make the distributed simulation to become worse in performance than the single-node simulation.

Our recent research dealt with the techniques for minimizing the impacts of the update-dependency among distributed simulator objects. Basic approaches were formulated [7] and experimental research is under way.

3.2 CAMIN testbed

One of the DTS applications developed in the DREAM Lab at UC Irvine is the *Coordinated Anti-Missile Interceptor Network (CAMIN)* simulation. The application scenario in this low-cost military C^3 (command, control, and communication) distributed computing simulation is that of detecting and intercepting hostile reentry vehicles to protect a commander ship. Army which operates radars, ground-based interceptor launchers, and a command-control center cooperates with Navy which operates the important commander ship with self-defense capabilities. Fighter airplanes are also used to launch airborne interceptors. The cooperative computing TMOs, the real-time simulator TMOs, and the programs for graphic display of the status of the application environment can be distributed over the LAN of one to five computing nodes.

In addition to the basic simulation of this military C3 distributed computing scenario, a fault tolerance feature called the *primary-shadow TMO replication (PSTR)* is accommodated into this simulation. A set of major components such as radar data queue (RDQ), flying object tracking (FOT), and intercept plan data structure (IPDS) TMOs are actively replicated to tolerate a logical or timing fault. Fig. 3 shows the configuration of CAMIN simulation in a four-node setup with replicas of the above three TMOs. Also, the *supervisor-based network surveillance (SNS)* scheme detects a failure of a primary node and notifies the associated shadow TMO of the failure of its primary so that the shadow can take over the job of the primary in a bounded time.

4. Conclusion

The TMO scheme for wide area DRC is promising, especially with the advent of a new-generation network infrastructure such as OptIPuter. Nevertheless, this field is in an early stage. The TMO-structured DTS has been demonstrated in reasonably convincing forms but its optimal use requires much further research.

Acknowledgment: The work reported here was supported in part by the NSF under Grant Numbers 02-04050 (NGS) and in part by the NSF under Cooperative Agreement ANI-0225642 to UCSD for "The OptIPuter".

References

[1] Jenks, S.F., et al., "A Linux-Based Implementation of a Middleware Model Supporting Time-Triggered Message-Triggered Objects", *Proc. 8th IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, Seattle, May, 2005, pp. 350-358.

[2] Kadavil, Anish, 'An enhanced GUI approach for programming real-time distributed systems with graphical interfaces', MS Thesis, EECS Dept., UCI, Dec. 2003.

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

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

[5] Kim, K.H., Ishida, M., and Liu, J., "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. ISORC '99 (IEEE CS 2nd Int'l Symp. on Object-oriented Real-time distributed Computing)*, May 1999, pp.54-63.

[6] Kim, K.H., "APIs for Real-Time Distributed Object Programming", *IEEE Computer*, June 2000, pp.72-80.

[7] 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.

[8] Kim, K.H., "Commanding and Reactive Control of Peripherals in the TMO Programming Scheme", *Proc. ISORC '02 (5th IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing)*, Crystal City, VA, April 2002, pp.448-456.

[9] 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.

[10] Kim, K.H., and Kang, S.J., "A GUI Approach to Programming of TMO Frames and Design of Real-Time Distributed Computing Software", *Proc. ISADS 2003 (IEEE CS 6th Int'l Symp. on Autonomous Decentralized Systems)*, Pisa, Italy, April 2003, pp.53-60.

[11] 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.

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

[13] Smarr, L.L., Chien, A.A., Defanti, T., Leigh, J., and Papadopoulos, P.M., "The OptIPuter", *Comm. ACM*, Nov. 2003, Vol. 46, No. 11, pp.59-67.