Lifting the Level of Abstraction Dealt with in Programming of Networked Embedded Computing Systems

K. H. (Kane) Kim

EECS Dept. & DREAM Lab.
University of California, Irvine
Irvine, CA 92697, USA
khkim@uci.edu
(Keynote)

Abstract[1] The scale and complexity of advanced networked embedded computing (NEC) application systems are steadily growing. The need has become increasingly acute for a programming method and style that imposes much less amounts of detail-handling on the real-time (RT) distributed computing (DC) programmer than the currently widely used method does. With this motivation a number of different attempts have been made toward establishing high-level RT DC objects. The TMO scheme developed over the past 18 years by the author and his collaborators is one of those attempts. In terms of lifting the level of abstractions of main program building-blocks, the TMO scheme has been about the most daring attempt. However, all the attempts have not yet reached the level of sufficient maturity in that the ease of guaranteeing the timeliness of critical output actions with a high degree of precision has not been much demonstrated. Some basic principles and techniques learned from past research on TMO are briefly reviewed. Then major remaining research issues are discussed.

Keywords *real time, distributed computing, networked embedded computing, TMO, time-trigger, message-trigger, object, timeliness, guarantee*

1. MOTIVATION

In recent years the scale and complexity of advanced networked embedded computing (NEC) application systems have been steadily growing. To cope with this growing complexity, the programming styles, methods, and tools need to be significantly upgraded.

The predominant style of programming used today in the young field of NEC application programming is the low-level programming that can be called the *thread - UDP – thread-priority* (TUP) programming. For the past two decades I have belonged to the small community which strongly believe in the need for upgrading the level of abstractions of the program building-blocks available to real-time (RT) distributed computing (DC) programmers. The fundamental problem with the TUP programming is that understandable specification, design, and timeliness guaranteeing is very difficult. To be more specific, the following limitations exist:

[1] The author is also an adjunct faculty member of Konkuk University.

(Pr1) The quality of advanced NEC software produced by use of the TUP approach tends to be low due to the difficulty of obtaining understandable designs.
(Pr2) The productivity of NEC software engineers is low due to the need for them to deal with tedious details, which can be avoided when using a higher-level programming approach, as well as the difficulty of obtaining understandable designs.
(Pr3) The performance of advanced NEC software tends to be low due to the difficulty of understanding the performance impacts of detailed design choices and achieving optimizations. The relationship between the application requirements, especially *timing requirements*, and the designs of TUP programs is very difficult to trace. As a result, ineffective use of resources in NEC systems results.

Therefore, the need has become increasingly acute for a programming method and style that imposes much less amounts of detail-handling on the RT DC programmer. This means specifically the following:

(Up1) A higher-level RT DC program model consisting of program structures and vocabularies that are more abstract and yet do not force compromises in the degree of control of various important action timings, is desired.
(Up2) A desirable program model should exhibit more directly and clearly the relationship between itself and the application requirements that it is intended to meet, including the timing and other performance requirements.
(Up3) As the core part of the desirable program model being sought, the program construct which can play the role of the *main building-block* in development of RT DC application programs was singled out from the beginning phase of the research on high-level RT DC programming. In addition, given the insufficient understanding that had existed in the field regarding program constructs and the desire to lift the level of abstraction which RT DC programmers would need to deal with without removing any important kind of programming power from them, the small community quickly reached a conclusion that the main building-block should be some version of the abstract data type *object*. Therefore, abstract RT DC object models have been at the top in the list of research targets in this area.

Among a number of different attempts toward establishing such high-level RT DC objects, the efforts that have been pursued with the greatest persistence are the following three: RT Corba [OMG05], RT Java [Bru09], and TMO (*Time-triggered Message-triggered Object*) [Kim97, Kim00, Liu09]. In terms of lifting the level of abstractions of main program building-blocks, the TMO scheme can be viewed as the most daring and advanced attempt. However, all three attempts have not yet reached the level of sufficient maturity in that the ease of guaranteeing the timeliness of critical output actions with a high degree of precision, e.g., sub-millisecond-level guaranteeing of a result return from a remote object-method invocation, has not been sufficiently demonstrated. Much further research remains to be done.

In the next section, some basic principles and techniques learned from past research on establishing the high-level RT DC object, TMO, are briefly reviewed. Then major remaining research issues are discussed in Section 3.

2. PRINCIPLES AND APPROACHES NEWLY EXPLORED IN TMO RESEARCH

In 1992 the author decided to adopt a simple skeleton model with concrete syntactic structure and semantics, which was initially named the RTO.k and later renamed to TMO (*Time-triggered Message-triggered Object*) [Kim97, Kim00, Liu09]. In the past 18 years, the progress in fully developing and maturing the TMO programming technology has been slow but steady. In fact, all efforts geared toward establishing high-level RT DC objects have been advancing at similar rates. Some parts of the TMO model were concrete mechanizations of the basic principle of *global-Time based Coordination of Distributed computing Actions* (TCoDA), which had been first advocated by Hermann Kopetz in Austria [Kop87, Kop97].

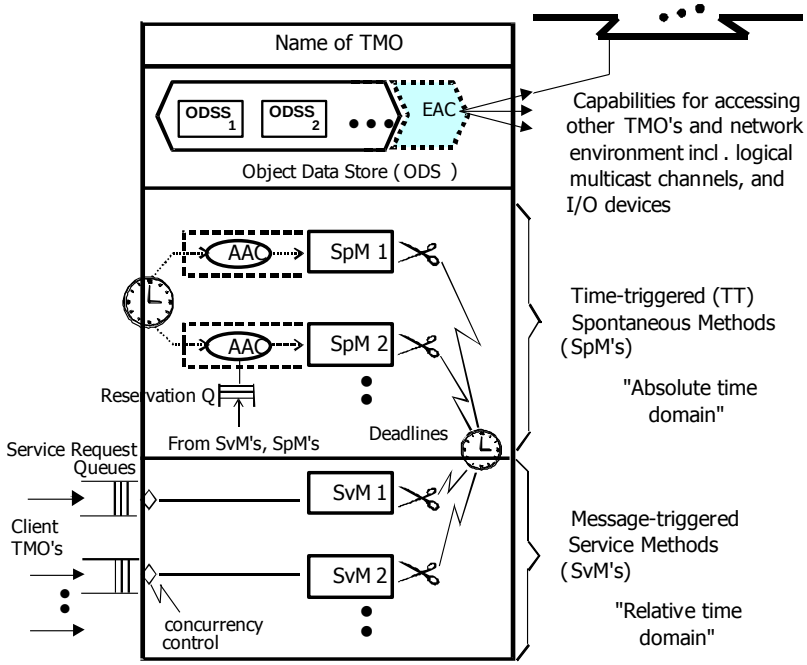

Figure 1. Structure of the TMO (adapted from [Kim97])

The basic TMO structure is depicted in Figure 1. Calling the TMO scheme a high-level DC 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.

(3) *Timing requirements need to be specified only in the most natural form of 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.

The program structuring principles and approaches newly explored in TMO research are summarized in this section. These principles and approaches are considered to be of fundamental nature and useful in practical NEC programming and software engineering.

2.1 Pervasive use of a global time base

Practically all time references in a TMO are references to *global time* [Kop97] in that their meaning and correctness are unaffected by the location of the TMO. If GPS receivers are incorporated into the TMO execution engine, then a global time base of microsecond-level precision can be established easily. Within a cluster computer or a LAN based DC system a master-slave scheme, which involves time announcements by the master and exploitation of the knowledge on the message delay between the master and the slave, can be used to establish a global time base of 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.

This pervasive use of a global time base in networks of RT DC objects was first explored in the TMO scheme and is regarded as a fundamental approach of high usefulness in advanced NEC software engineering.

2.2 Deadline for result arrival (DRA)

TMO is a natural, syntactically minor, and semantically powerful extension of the conventional object(s). TMO is a DC component and thus TMOs distributed over multiple nodes may interact via *remote method calls* and another mechanism discussed below in (G). Object-methods in a TMO that can be called from other TMOs are called message-triggered *service methods* (SvMs).

It takes two simple statements to construct a remote method call:

```
(Stmt1) SvMGateClass Gate1 (_T("TMO3“), _T("SvM7“),
    tm4_DCS_age(7*1000*1000) )
(Stmt2) Gate1.BlockingSR ( void* ParamPtr, int ParamSize, MicroSec DRA, tms
    ORT )
```

The first statement Stmt1 is for instantiation of a proxy object, called a *TMO-method gate*, for the remote TMO-method (TMO3.SvM7) to be invoked. This statement is placed in the *environment access capability* (EAC) *section* which is one of the four major parts of the TMO structure.

The other statement Stmt2 is a call for a built-in method of the TMO-method gate, BlockingSR(), which in turn performs a blocking type of a service request to the remote TMO-method.

The TMO-method gate possesses built-in methods for both blocking types and non-blocking types of service requests.

Note that the 3rd parameter in Stmt2 is a *deadline for return result arrival* (DRA). The delay from the instant at which this statement is executed to the instant at which the result returned from the invoked remote SvM becomes available in the node hosting the client should not exceed the DRA specified. Otherwise, the TMO execution engine which is spread over the network of computing nodes forming the DC network raises an error signal. In order to specify the DRA appropriately the TMO programmer needs to have good understanding of at least the *worst-case service time* of the remote SvM or a tight upper bound on the service time. In fact, each SvM includes an explicit specification of a *guaranteed execution duration bound* (GEDB) and a *maximum invocation rate* (MIR) supported by it.

The TMO was the first RT DC object / component which incorporated DRA as an integral mechanized part.

2.3 Object-data-store segment (ODSS) and concurrency control

One of the four major parts of the TMO structure is the *object-data-store* (ODS) *section* which contains the data-container variables shared among methods of a TMO. Variable are grouped into *ODS segments* (ODSSs), which are the units that can be locked for exclusive use by a TMO-method in execution. Access rights of TMO-methods for ODSSs are explicitly specified and registered to the execution engine which in turn analyzes them to exploit maximal concurrency. An execution of a TMO-method is launched only when all the ODSSs for which the TMO-method has access rights have been locked for use in the TMO-method-execution. Conversely, multiple TMO-method-executions may progress concurrently if for each of those executions the TMO execution engine has locked all the ODSSs needed for that method-execution.

This can be viewed as an approach for "maximal exploitation of concurrency among object-methods". TMO is among the earliest RT DC objects which incorporated such a concurrency control approach.

2.4 Spontaneous method (SpM)

TMO is 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". All time references here are global-time references. By using SpMs, the TCoDA principle can be easily designed and realized.

If an SpM has the specification of triggering times, "for t = from system-startup to system-shutdown every eternity start-during (t, t + minimum-activation-delay) finish-by system-shutdown", the function body is executed from the system-startup instant until the system-shutdown instant or the completion of the function body, whichever comes first. Therefore, such SpM can be viewed as a "thread" activated upon system startup to run the function body expressed in a high-level language form.

The logic in such function body may be designed to check the global time and sleep until the global time reaches a certain time-point. It may also access a certain pointer and invoke a local function pointed to by that pointer.

The SpM as an object-method was first explored in the TMO scheme. The SpM is regarded as a fundamental approach of high usefulness in advanced NEC software engineering.

2.5 Basic concurrency constraint (BCC)

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. 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.

Under BCC the timing behavior of SpMs is not impacted in complicated ways by the SvM executions, especially if the CPU sharing among independent SpM executions and SvM executions is handled such that the share of the CPU time each SpM execution receives can be easily calculated. Therefore, the analysis of the timing behavior of a TMO can proceed largely in two steps, i.e., analyze the *execution time bounds*, which can also be viewed as the *service time bounds* (STBs), of SpMs first and then analyze the STBs of SvMs. The STBs of SvMs tend to be less tight than the STBs of SpMs.

BCC was first explored in the TMO scheme. The author believes that this BCC is also a fundamental approach of high usefulness in advanced NEC software engineering.

2.6 Ordered isolation (OI) rule

The difficulty of analyzing competitions among method-executions depends much on the way ODSSs are locked and released. To reduce that difficulty further after incorporating BCC, the TMO scheme adopted the *ordered isolation* (OI) rule [Kim07b]. The OI rule can be stated by using the term *initiation timestamp* (*I-timestamp*) defined as follows:

- In the case of an SvM execution, the *I-timestamp* is defined as the record of the time instant at which the execution engine initiated the SvM execution after receiving the client request and ensuring that the SvM execution can be initiated without violating BCC.
- In the case of an SpM execution, the *I-timestamp* is defined as the record of the time instant at which the SpM execution was initiated according to the AAC specification of the SpM.

The OI rule has the following two parts:

(OI-1) A method-execution with an older *I-timestamp* must not be waiting for the release of an ODSS held by a method-execution with a younger *I-timestamp*.

(OI-2) A method execution must not be rolled back due to an ODSS conflict.

If these rules or other rules restricting the possible types of waiting situations and rollback situations are not followed, then the validation of GEDBs can be drastically more complicated. The price paid for reducing the complexity of deriving tight execution duration bounds by adopting the OI rule is the loss of some concurrency.

The OI rule was first explored in the TMO scheme. A rule that allows a greater amount of concurrency than the OI rule does and yet does not make the derivation of tight execution duration bounds of TMO-methods is an open research topic.

2.7 Real-time Multicast and Memory replication Channel (RMMC)

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) [Kim00, Kim05]. The RMMC scheme facilitates RT publish-subscribe channels in a powerful form. It supports not only conventional *event messages* but also *state messages* based on distributed replicated memory semantics [Kop97].

The access gates are called *RMMC gates* and treated as special types of ODSSs. Access rights of TMO-methods for RMMC gates are thus explicitly specified and registered to the execution engine. RMMC gates are declared in the EAC section. The declaration of an RMMC gate includes a parameter specifying a *bound on the message transmission delay over the channel*.

When a message is multicast over an RMMC by calling for a built-in method of the corresponding RMMC gate, an *official release time* (ORT) is tagged to the message. When the message reaches a computing node hosting a subscriber TMO, the message cannot be opened until the ORT arrives. This ORT mechanism is useful in synchronizing message-pickups by multiple subscribers or ordering message-pickups by a subscriber of the messages coming over multiple RMMCs to the same TMO. The ORT is also incorporated into remote method call mechanisms.

The part of the RMMC that carries event messages is an extension of the data field scheme initiated by Hitachi, Ltd. in Japan [Kim95, Mor93] and the part that carries state messages is an extension of the messaging scheme initiated by Hermann Kopetz. The ORT idea was initiated by Toshiba Corp., Japan. The following four concepts or approaches were first explored in the TMO scheme.

(1) The use of an RT logical multicast channel as a mechanism for interconnecting RT DC objects.
(2) A single RT logical multicast channel that carries both event messages and state messages.
(3) The approach of using an RMMC gate in connecting an RT DC object to an RT logical multicast channel.
(4) The incorporation of the ORT into multicasts of RT event messages and state messages.

Through the TMO programming experiences over the years the author has become convinced that this RMMC is a fundamental mechanism of high usefulness in advanced NEC software engineering.

2.8 Underground Non-Blocking Buffer (NBB) with a pair of access gates

In the basic TMO scheme, there is no way for any two concurrent method executions to exchange any data. This is because an ODSS cannot be shared when at least one method-execution needs to access it in the *read-write* mode.

In some application cases, it is desirable to let two long-running SpM executions exchange data streams. The TMO was recently extended by incorporating a new mechanism that enables multiple rounds of data passing from one method-execution to the other method-execution and yet does not damage the nature of the TMO scheme which makes STB validation relatively easy. The mechanism is based on the *Non-Blocking Buffer* (NBB) developed in recent years by several teams [Var01; Kim06; Kim07a]. An NBB used between a producer thread and a consumer thread allows the producer to insert a new data item into its internal circular buffer at any time without experiencing any blocking. If the internal circular buffer is saturated, then the producer attempting to insert a new item can detect it immediately and choose to do other things for a while and then check the NBB again. Similarly, the NBB allows the consumer to retrieve a data item from the internal circular buffer at any time without experiencing any blocking.

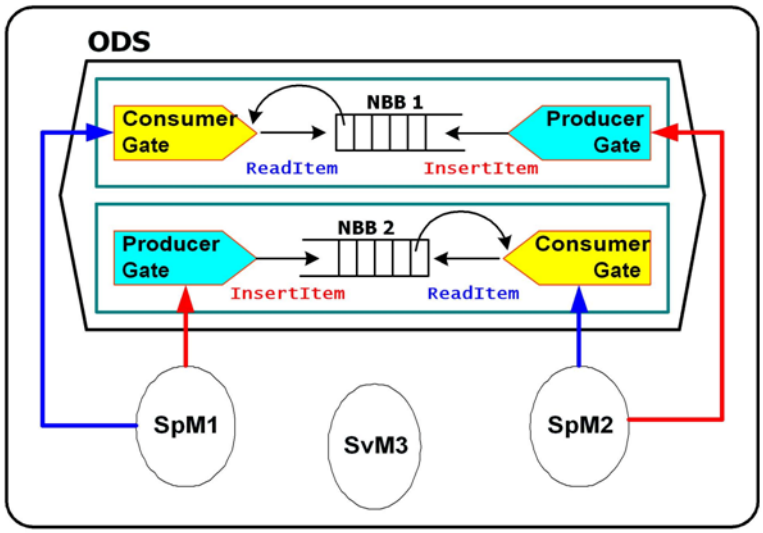

Figure 2. Underground NBBs in a TMO (Adapted from [Kim07b])

The version of NBB that is appropriate for use between two methods in a TMO is depicted in Figure 2. Two NBBs are there. Each NBB consists of an internal circular buffer, a *producer gate*, and a *consumer gate*. The two gates are ODSSs and they are registered with the execution engine. In a sense, the internal circular buffer is treated as an invisible data structure. Therefore, the producer method puts a read-write lock on the producer gate and the consumer method puts a read-write lock on the consumer gate and then the two are treated by the execution engine as two independent methods not

sharing any data structure. Only the TMO designer knows that the internal circular buffer is a shared data structure but the execution engine *pretends* not to know this and allows the producer method-execution and the consumer to proceed concurrently. Therefore, this version of NBB is called the *underground NBB*.

The author believes that the NBB is an important mechanism for use in RT concurrent programs and TMOs containing underground NBBs may be the first instance of using NBBs in RT DC programs.

3. MAJOR REMAINING RESEARCH ISSUES

Over the past 15 years TMO execution facilities have been continuously enhanced along with the related APIs (application programming interfaces). A middleware model called the TMOSM (*TMO Support Middleware*) provides execution support mechanisms and can be easily adapted to a variety of commercial kernel-hardware platforms in wide use in industry [Kim99; Jen07, Kim08]. Therefore, a TMO execution engine consists of a group of networked computing node platforms (hardware nodes, plus OS kernels) and instantiations of the *TMO Support Middleware* (TMOSM) running on the node platforms. TMOSM uses well-established services of commercial OS kernels, 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 of TMOSM currently exist for Windows XP / Vista, Windows CE, and Linux 2.6 platforms.

Along with TMOSM, the *TMO Support Library* (TMOSL) has been developed [Kim99; Kim00; Kim05]. It provides a set of friendly application programming interfaces (APIs) that wrap the execution support services of TMOSM. TMOSL defines a number of C++ classes and enables convenient high-level programming by approximating a programming language directly supporting TMO as a basic building block. Other research teams have also developed TMO execution engines based on different kernel platforms [KimH02; KimJ05].

TMOSM, TMOSL, and other tools are available for down-load from web, http://dream.eng.uci.edu /tmodownload/.

A number of RT DC applications have also been developed by use of the TMO scheme [Hen08, Liu09]. Some demo applications can be seen in http://dream.eng.uci.edu/demo.

However, there are a number of issues that need to be resolved through further research before the technology can reach the level of full maturity. Some major issues are listed below.

3.1 Intermediate-level RT DC programming scheme

For implementing NEC applications on small computing platforms, the DC object approach such as the TMO scheme could appear overhead-heavy. Also, transitioning from the TUP programming to a high-level programming approach such as TMO programming may be too big an adjustment in many industrial environments. Therefore, there seems to be a good justification for establishing a complimentary RT DC programming scheme which is based on a somewhat lower level of abstractions of program building-blocks.

I believe that one promising direction is to take the object framework off the TMO and let SpMs become independent *time-triggered functions* (TTFs) and SvMs become *independently threaded service functions* (ITSFs). Then the resulting programming approach is to build RT DC programs with TTFs, ITSFs, NBBs, logical multicast channels, and other data sharing mechanisms. Such *intermediate-level programming* approach still avoids the direct use of threads and tread-priorities.

In order to avoid creating overhead-heavy environments, it will be necessary to develop "kernel-level support" for building-blocks of such intermediate-level RT DC programs.

3.2 Service time bound (STB*)*

Although the TMO scheme was initiated with the goal of enabling timeliness guaranteeing, only very limited experimental research on deriving STBs of TMO-methods has been performed [Col09]. This is the primary reason why the TMO scheme, and any other RT DC programming scheme for that matter, cannot be said to have reached a sufficiently mature level. The extent of research conducted all over the world in that direction, i.e., toward establishing a sound technical foundation for derivation of tight STBs of RT DC programs, looks quite unimpressive at present. Considering the critical needs for such research, the author hopes that the situation change soon.

To establish a useful technical foundation, much better understanding needs to be obtained on issues such as allocation of both computing and communication resources to concurrent method-executions, allocation of resources in manners driven by quality-of-service specifications (e.g., timing specifications in TMOs), utilization of multi-core CPUs, timeliness-guaranteed handling of I/O activities, etc.

ACKNOWLEDGEMENTS

The research reported here has been supported over the years by a number of sponsors, e.g., NSF, DARPA, ARO, Microsoft Corp., Hitachi Ltd., ETRI, and the WCU program in Konkuk University sponsored by MEST, Korea. No part of this paper represents the views and opinions of the sponsors mentioned above.

REFERENCES

1. Eric J. Bruno and Gregory Bollella, Real-Time Java Programming: With Java RTS, Bru09, Sun Microsystems, (2009).
2. J.A. Colmenares, K.H. (Kane) Kim, and D.H. Kim, Experimental Evaluation of a Hybrid Approach for Deriving Service-time Bounds of Methods in Real-time Distributed Computing Objects, *Proc. IESS '09 (*Int'l Embedded Systems Symp., Langenargen, Germany) (2009).
3. Emmanuel Henrich et al., Realization of an Adaptive Distributed Sound System Based on Global-time-based Coordination and Listener Localization, Proc. ISORC 2008 (11th IEEE CS Int'l Symp. On Object/Component/Service-Oriented Real-time distributed Computing), Orlando, FL, May (2008) pp.91-99.

4. S.F. Jenks, K.H. (Kane) Kim, et al., A Middleware Model Supporting Time-Triggered Message-Triggered Objects for Standard Linux Systems, Real-Time Systems - The International Journal of Time-Critical Computing Systems, Vol. 36, No.1, April (2007), pp.75–99.

5. K.H. Kim, K. Mori, and H. Nakanishi, Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HU-DF Inter-Process-Group Communication Scheme, Proc. IEEE 2nd Int'l Symp. on Autonomous Decentralized Systems (ISADS '95), Phoenix, AZ, April. (1995), pp.305-312.

6. K.H. Kim, Object structures for real-time systems and simulators, IEEE Computer, 30(8):62–70, Aug. (1997).

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

8. K.H. Kim, APIs for Real-Time Distributed Object Programming, IEEE Computer, 33(6):72-80, June (2000).

9. K.H. Kim, Y.Q. Li, S. Liu, et al., RMMC Programming Model and Support Execution Engine in the TMO Programming Scheme, Proc. ISORC 2005 (8th IEEE CS Int' Symp. On Object-Oriented Real-Time Distributed Computing) , May, (2005), pp.34- 43.

10. K.H. Kim, A Non-Blocking Buffer Mechanism for Real-Time Event Message Communication, Real-Time Systems 32, 3 (2006), pp.197-211.

11. K.H. Kim, J.A. Colmenares, and K.W. Rim, Efficient Adaptations of the Non-blocking Buffer for Event Message Communication between Real-Time Threads, Proc. ISORC 2007 (10th IEEE CS Int'l Symp. on Object & Component Oriented Real-Time Distributed Computing), Santorini, Greece, May, (2007), pp.29-40.

12. Kim, K.H., and Colmenares, J., Maximizing Concurrency and Analyzable Timing Behavior in Component-Oriented Real-Time Distributed Computing Application Systems, KIISE Journal of Computing Science and Engineering (JCSE), Vol.1, No.1, Sep. (2007), pp. 56-73 (downloadable from http://jcse.kiise.org/).

13. K.H. Kim, Y.Q. Li, K. W. Rim, E. Shokri, A Hierarchical Resource Management Scheme Enabled by the TMO Programming Scheme, Proc. ISORC 2008 (11th IEEE CS Int'l Symp. On Object/Component/Service-Oriented Real-time distributed Computing), Orlando, FL, May (2008), pp. 370-376.

14. H.J. Kim, S.H. Park, J.G. Kim, M.H. Kim, and K.W. Rim, TMO-Linux: A Linux-based real-time operating system supporting execution of TMOs, Proc. 5th IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2002), pp. 288-294.

15. J.G. Kim, M. Kim, K. Kim, and S. Heu, TMO-eCos: An eCos-Based Real-Time Micro-Operating System Supporting Execution of a TMO StructuredProgram, Proc. 8th IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2005), pp.182-189.

16. H. Kopetz, and W. Ochsenreiter, Clock Synchronisation in Distributed Real-Time Systems. IEEE Trans. Computers, (1987). pp. 933-940.

17. Kopetz, H., Real-Time Systems: Design Principles for Distributed Embedded Application, Kluwer Academic Pub., ISBN: 0-7923-9894-7, Boston, (1997).

18. Sheng Liu, K. H. Kim, Zhen Zhang, S.P. Lee, and K.W. Rim, Achieving High-Level QoS in Multi-Party Video-Conferencing Systems via Exploitation of Global Time, Proc. ISORC 2009 (12th IEEE CS Int'l Symp. on Object/Component/Service-Oriented Real-time distributed Computing), Tokyo, Japan, Mar. (2009)

19. Mori, K., Autonomous Decentralized Systems: Concept, Data Field Architecture, and Future Trends, Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 93), Mar. (1993), pp. 28-34.

20. Object Management Group, Real-time CORBA Specification, Version 1.2', (formal/05-01-04), Jan. 2005. Available at: http://www.omg.org/cgi-bin/apps/doc? formal/05-01-04.pdf

21. P. Varma, Two Lock-Free, Constant-Space, Multiple-(Impure)-Reader, Single-Writer Structures, US Patent No. 6304924 B1, (2001).