

## Questionable Relevancy of Fixed Priority Assignment in Distributed Object Design

**K. H. (Kane) Kim**

University of California  
Irvine, CA 92697, U.S.A.  
Kane@Ece.Uci.Edu  
(Position paper)

The following fundamental issues in establishing sound architectures for object-oriented (OO) real-time (RT) distributed computing systems (DCS's) are briefly discussed in this paper:

- (1) How much is the fixed priority assignment relevant to the activities of distributed object designers ?
- (2) Can the event-triggered architecture and the time-triggered architecture be combined into a unified architecture ?

### 1. Fixed priority assignment

Until recently, assigning fixed priorities to processes was an art frequently practised by designers of RT computing systems, especially simple centralized RT control systems. The temptation to continue this habit into the era of designing complex RT systems in the form of networks of location-transparent high-level distributed objects may exist. However, asking distributed object designers to handle fixed priority assignment is considered to be an ineffective approach that cannot be justified. Some of the major reasonings are as follows.

(1) One major goal of moving from process-structured RT distributed computing (DC) to OO RT DC is to reduce the complexity in design of large-scale RT application systems. The complexity reduction is achieved by allowing the system designer to deal with high-level abstract computation units, i.e., distributed objects, while leaving the details of mapping the high-level designs to machine execution resources to the tools and intelligent execution engines. For example, it is desirable to relieve the distributed object designer of all or most of the burden of knowing the site locations of distributed objects or the low-level protocols for inter-node communication.

(2) Fixed priorities are the attributes that can be easily observed by the low-level node execution engine. However, it is a very crude way of expressing the *urgency* of application processes. Except in simple cases of centralized systems, the designer does not clearly understand the relationship between the timing requirements of application processes and the timing behavior effected by assigning fixed priorities to the processes during the design period. A typical situation where the designer could assign fixed priorities with relatively less pain was in designing a simple device

control system in which the main processor runs multiple fixed-pattern periodic processes, each repeatedly exchanging signals/messages with one specifically assigned cyclic device in a fixed pattern without much interaction with other processes. Many large-scale RT DCS's needed nowadays are far different from such simple centralized systems. They run many cooperative dynamic decision-making application subsystems which exhibit dynamically changing patterns of interaction among themselves as well as with the application environments.

(3) If there are timing requirements inherent in the target applications, it is best for the distributed object designer to express them and reflect them in the simplest, clearest, and the most easily analyzable form in the high-level object network design. If the application requires a certain object method to start within a certain time-window and complete by a certain deadline, the start time-window and the completion deadline must be expressed in natural forms as parts of the distributed object design. Even without considering possible object migration and dynamic changes in the network configurations and object distributions, it will be a horrible task for the distributed object designer to assign fixed priorities to the objects or their methods with the hope that they will effect the desired start time-windows and completion deadlines.

Suppose object O1 was designed to contain two methods, O1.m1 and O1.m2, and object O2 was designed to contain two methods, O2.m3 and O2.m4. In order to meet the application requirements, O1.m1 must be completed in 10 milliseconds once it is activated. O1.m1 needs a service from the method in the third object, O3.m5, and the complete service must be obtained in 5 milliseconds in order to meet its own completion deadline of 10 milliseconds. If O1.m2, O2.m3, and O2.m4 have similar requirements, how can one assign fixed priorities to all these methods ? What confidence does the designer have on the effects of such an assignment ? If the distributed object designer must do this kind of activities, then most of the high-level abstract nature of the object network design activities disappears. Therefore, distributed object designers who produce location-transparent distributed objects which make remote method calls among themselves should not think in unnatural terms such as priorities.

(4) Given natural expressions of timing requirements that the designer has embedded in the designs of

distributed objects, it should be the job of the middleware, the operating system, and the design tools to map the designs to low-level node execution engines managing low-level resources such as processes and threads regardless of whether the low-level resources are assigned fixed priorities or dynamic priorities.

## 2. Unification of the event-triggered architecture and the time-triggered architecture

The complimentary relationship between the event-triggered (ET) architecture and the time-triggered (TT) architecture have been a vigorous discussion topic in the past decade [Kop90, Kop97, Lei98]. One safe conclusion is that the ET architecture may lead to a better economy of scale for certain kinds of RT applications whereas it may be easier with the TT architecture to realize more predictable timing behavior and more dependable behavior of systems for certain kinds of hard RT applications. It thus seems to be a meaningful research action to pursue an architecture and a design methodology that encompass both ET and TT approaches.

One such approach being explored by the author and his research collaborators is the the *Time-Triggered Message-Triggered Object* (TMO) structuring scheme [Kim94, Kim97b]. The TMO scheme is intended 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 is a syntactically simple and natural but semantically powerful extension of the conventional object structuring approaches. The scheme facilitates uniform structuring of (1) both RT and non-RT distributed application systems, (2) both control computer systems and their application environment simulators, and (3) requirement specifications and system designs arising at various phases of system engineering.

At the same time the TMO scheme enables a great reduction of the designer's efforts in guaranteeing timely service capabilities of distributed computing application systems. Its support tools can be based on well-established OO programming languages such as C++ and JAVA and on ubiquitous commercial RT operating system kernels or even on NT. In addition, the TMO scheme facilitates an attractively simple approach to parallel and distributed RT simulation [Kim94, Kim97b].

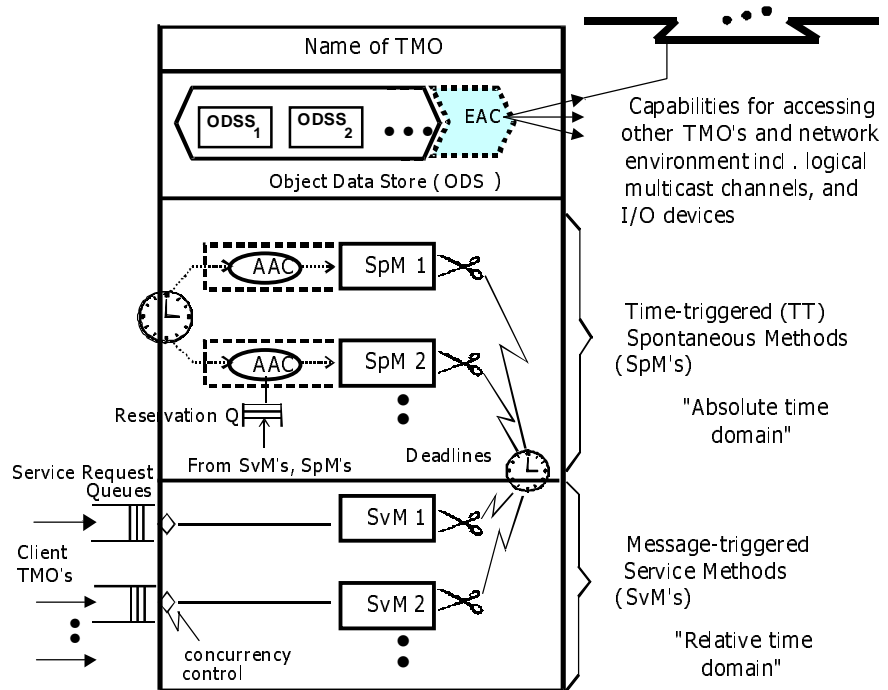


Figure 1: The basic TMO structure (adapted from [Kim97b])

TMO extends the conventional objects with several unique features including:

(TF1) Spontaneous method (SpM):

The TMO contains a new type of methods, *time-triggered (TT-) methods*, also called the *spontaneous methods* (SpM's), in addition to the conventional *service methods* (SvM's). The SpM executions are triggered when the RT clock reaches specific values determined at design time whereas the SvM executions are triggered by service request messages from clients. Moreover, actions to be taken at real times *that can be determined at the design time* can appear only in SpM's.

(TF2) *Basic concurrency constraint (BCC)*:

Under this rule, SvM's cannot disturb the executions of SpM's and the designer's efforts in guaranteeing timely service capabilities of TMO's are greatly simplified. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place*. An SvM is allowed to execute only when no SpM that accesses the same portion of the Object Data Store (ODS) (i.e., a group of data members) to be accessed by this SvM has an execution time window that will overlap with the execution time window of this SvM. However, the BCC does not stand in the way of either concurrent SpM executions or concurrent SvM executions.

(TF3) A *time window* imposed on each output action and completion of a method:

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 object. Before determining the time window specifications, the server object designer must make sure that with the available *object execution engine* (hardware plus operating system) the server object can be implemented such that the output actions are performed within the time windows.

(TF4) *Programmable data-field-channels* [Kim95, Kim97a].

This feature of the TMO scheme facilitates highly abstract (relieving the programmer of the burden of dealing with underlying OS services and network protocols) and yet highly efficient cooperation among remote TMO's. The essence of the original data field scheme [Mor93] is to facilitate dynamic creation of *logical multicast channels* shared among the concurrent distributed processes for their interaction in such a way that the idiosyncracies of the physical communication networks are transparent to the process designer. If the physical communication facility has the broadcast capability, then a logical multicast channel is facilitated by making all processing nodes using the channel broadcast (through the physical communication facility) messages with the headers containing the ID of the channel called the *content code*. The processing nodes connected to the logical channel can see all the messages coming through the physical broadcast facility but will pay attention only to those messages containing relevant content codes. This means that when processes are designed to communicate via the logical multicast channels only, processes can be dynamically relocated without impacting other cooperating processes. The programmable DFC scheme [Kim95, Kim97a] differs from the original data field scheme in that the former allows dynamic flexible connection of processes to the logical channels and supports not only conventional *event messages* but also *state messages* [Kim95, Kop97] which are based on the distributed replicated memory semantics.

Several preliminary experimental validations undertaken by the proposers and other research organizations [Kim94, Kim97b, Sho98] have shown the great potential of the TMO structuring in design and implementation of large-scale complex distributed applications in factory automation, traffic control, and military planning/engagement areas. These demonstrations included successful implementations of augmented C++ runtime environments [Kim97a, Sho98]. However, much further work is needed on *software engineering environments* (SEE's) for TMO-structured distributed software in order to realize its main goal, i.e., to bring about a major improvement in the design productivity and product reliability in complex distributed system engineering.

**Acknowledgment:** The research work reported here was supported in part by the US Defense Advanced Research Project Agency under Contract N66001-97-C-8516 monitored by SPAWAR.

## References

- [Kim94] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Oct. 1994, Dana Point, pp.36-45.
- [Kim95] Kim, K.H., Mori, K., and Nakanishi, H., "Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HUF Inter-Process-Group Communication Scheme", *Proc. 1995 IEEE CS's 2nd Int'l Symp. on Autonomous Decentralized (ISADS 95)*, Phoenix, AZ, April. 1995, pp.305-312.
- [Kim97a] Kim, K.H., Subbaraman, C., and Bacellar, L., "Support for RTO.k Object Structured Programming in C++", *Control Engineering Practice*, an IFAC journal, Vol.5, No.7, 1997, pp.983-991.
- [Kim97b] Kim, K. H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No. 8, August 1997, pp. 62-70.
- [Kop90] Kopetz, H. and Kim, K.H., "Temporal Uncertainties in Interaction among Real-Time Objects", *Proc. IEEE Computer Society's 9th Symp. on Reliable Distributed Systems*, Huntsville, AL, Oct. 1990, pp.165-174.
- [Kop97] Kopetz, H., *'Real-Time Systems'*, Kluwer Academic Pub., 1997.
- [Lel98] LeLann, G., "Proof-Based System Engineering and Embedded Systems", Lecture Notes in Computer Science, 1494, G. Rozenberg and F. Vaandrager eds., Springer-Verlag, Oct. 1998, pp. 208-248.
- [Mor93] 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, Kawasaki, Japan, pp. 28-34.
- [Sho98] Shokri, E., Crane, P., and Kim, K.H., "An Implementation Model for Time-Triggered Message-Triggered Object Support Mechanisms in CORBA-Compliant COTS Platforms", *Proc. IEEE 1st Int'l Symp. on Object-oriented Real-time dependable Computing (ISORC)*, Kyoto, Japan, April 1998, pp. 12-21.

# Proceedings

## 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99)

2-5 May 1999

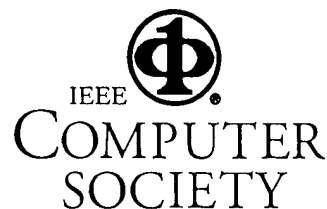
Saint-Malo, France

*Sponsored by*

IEEE Computer Society Technical Committee  
on Distributed Processing

*In cooperation with*

IFIP Working Group 10.4  
OMG  
INRIA



Los Alamitos, California  
Washington • Brussels • Tokyo

---