

## TMO-Based Programming in COTS Software/Hardware Platforms: A Case Study

Eltefaat Shokri

SoHaR Inc., Beverly Hills, CA

Kane Kim

University of California, Irvine, CA

### Abstract

*Object-oriented analysis and design methodologies have become popular in development of non-real-time business data processing applications. However, conventional object-oriented techniques have had minimal impact on development of real-time applications mainly because these techniques do not explicitly address key characteristics of real-time systems, in particular, timing requirements. The Time-triggered Message-triggered Object (TMO) structuring is in our view the most natural extension of the object-oriented design and implementation techniques which allows the system designer to explicitly specify timing characteristics of data and function components of an object.*

*To facilitate TMO-based design of real-time systems in the most cost-effective manner, we have developed a middleware (named TMOSM/ORB) providing TMO execution support mechanisms on top of the Windows NT operating system and a CORBA compliant object request broker. In order to evaluate the effectiveness of CORBA-compliant TMO based system development, a defense command-control application was ported into the TMOSM/ORB environment.*

*In this paper, first the basics of the CORBA-compliant TMO structuring scheme are presented. We then report the porting experience and its findings regarding the effectiveness of the CORBA-compliant TMO based programming in developing real-time applications.*

### 1. Introduction

Conventional object-oriented techniques [Boo94] do not explicitly address key characteristics of real-time systems, in particular, timing requirements. The Time-triggered Message-triggered Object (TMO) structuring [Kim94a, Kim97b] is an extension of the object-oriented design and implementation techniques that allows the system designer to explicitly specify timing characteristics of data and function components of an object. To facilitate design and execution of TMO-structured real-time systems in the most cost-effective manner, we

developed a middleware called TMOSM (TMO Support Middleware) [Sho98a, Sho98b] that provides execution support mechanisms and can be easily adapted to a variety of commercial software/hardware platforms compliant with industry standards. TMOSM/OmniORB2/NT (denoted by TMOSM/ORB hereafter) is an adaptation of TMOSM to the Microsoft Windows NT platform equipped with the public domain ORB (Object Request Broker), OmniORB2 [Lo98]. TMOSM/ORB has the following main characteristics:

- *Use of COTS software/hardware platforms:* TMOSM/ORB uses common features of modern commercial-off-the-shelf (COTS) platforms although it takes advantage of some advanced features of Windows NT, such as the real-time thread mechanism, in order to increase stability in the timing behavior of TMO-structured applications.
- *Compliance to CORBA standards:* CORBA standards [OMG95] provide a high-level abstraction for location-transparent language-neutral inter-object communications. TMO's supported by the middleware rely fully on standard ORB's for all of their inter-object communication needs.
- *Ease of programming and CORBA-compliant API:* The TMO was created with the clear goal of making it easy for conventional object oriented (OO) business application software engineers to accommodate. TMO programming with TMOSM/ORB is a minor extension of CORBA programming. In other words, a programmer familiar with CORBA programming needs to learn only a few additional system-calls (specific for handling timing features of TMO's). To further ease the TMO programming, the middleware provides a class library that encapsulates generic features and internals of the TMO support facility.
- *Structuring the application as a network of real-time objects:* TMOSM/ORB allows the programmers to design the application as a network of interacting TMO's. Also, the TMO structure facilitates easily analyzable multi-level representation of not only real-time computations but also distributed computing

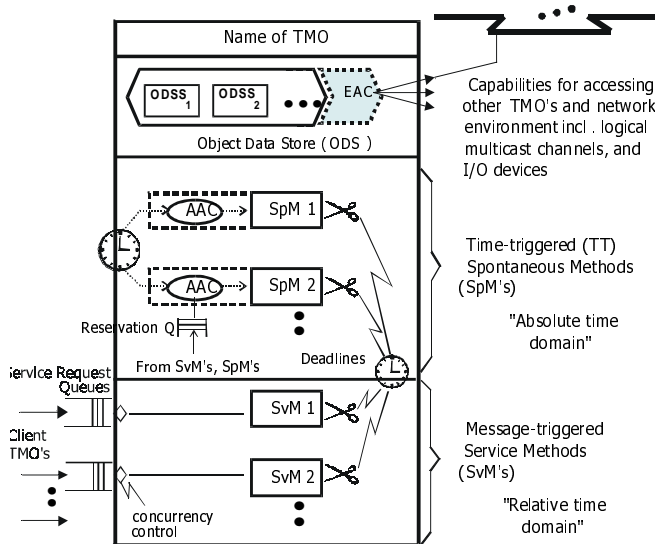
resources and application environments. Therefore, it enables creation of a network of TMO's that represent the computing application universe uniformly and rigorously.

There have been several evaluation and demonstration efforts on the effectiveness of TMO structuring in various application domains [Kim94b, Kim97c, Ngu97]. In this paper, we report an effort on porting a defense command-control application to the TMOSM/ORB environment. A brief discussion on CORBA-compliant TMO structuring is given in Section 2. Section 3 presents the experience on developing a real-time application on top of TMOSM/ORB. Section 4 provides a conclusion for the paper.

## 2. CORBA-compliant TMO structuring

### 2.1 TMO structuring

The Time-triggered Message-triggered Object (TMO) was established a few years ago [Kim94a, Kim97a] with a concrete syntactic structure and execution semantics for design and implementation of real-time systems. TMO is depicted in Figure 1 with some of its major characteristics as follows:



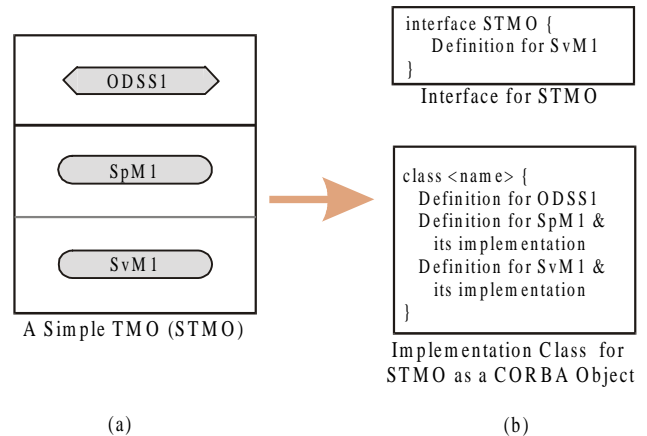
**Figure 1: The TMO Structure**

- Time-triggered methods (called spontaneous methods or SpM's) may be provided and must be clearly separated from conventional methods (called service methods or SvM's) triggered by client requests.
- Each SpM is associated with an autonomous activation condition (AAC) that specifies the times at which the associated method should be activated.
- Each method (SpM or SvM) is associated with a completion deadline and output action deadlines.

### 2.2 CORBA-compliant TMO structuring

An easy-to-use API was the main objective in designing support for execution of CORBA-compliant TMO's. The approach adopted here maps each TMO into a single CORBA object. The programmers who are familiar with CORBA-compliant implementation and programming principles will then need to learn only a few system-calls before structuring the applications into TMO networks. These system-calls are for TMO-specific features such as (i) distinguishing time-triggered methods from message-triggered methods and (ii) specifying timing characteristics of each TMO.

We adopted a simple mapping (S-mapping) that does not require extension of the CORBA IDL and yet covers a major portion of TMO applications. Under the S-mapping, externally seen elements of a TMO (i.e., SvM's) are defined in the object interface, while data elements and SpM's are defined only in the implementation of the CORBA object. Figure 2 illustrates the mapping of a simple TMO (with a single ODS segment, one SvM, and one SpM) in Figure 2 (a) into a CORBA object in Figure 2 (b). The Basic rules for the S-mapping are the following:



**Figure 2: S-Mapping of a TMO into a CORBA Object**

- A TMO is mapped into a single application-level CORBA object.
- Each SvM is represented in the object interface definition as an operation so that the client objects may see and call the SvM.
- Each SpM is viewed as a private method of the CORBA object and thus not represented in the object interface.
- Each object data store segment (ODSS) is viewed as a contained object and thus not represented in the object interface.

To further simplify the implementation of CORBA-compliant TMO's, common functions such as "registering TMO methods with the support manager" and "timely

association of threads to TMO methods", are isolated from the application-specific code produced by the application developer and are localized in a base class named TMOBaseClass. In other words, TMOBaseClass encapsulates the basic TMO management functions in a single base class such that the designer of the application TMO's need not be preoccupied with the details of basic TMO management issues, such as registering methods and assigning methods to kernel threads. A CORBA-compliant application TMO is thus an instance of a class derived from the TMOBaseClass class. Also, a CORBA-compliant TMO may have to be derived from some proxy classes depending on the ORB used, as shown in Figure 3.

To encapsulate the generic functionality of ODSS, another base class named ODSSBaseClass is provided from which each ODSS class is derived. TMOBaseClass and ODSSBaseClass are provided in the form of a library that can be used by the developer.

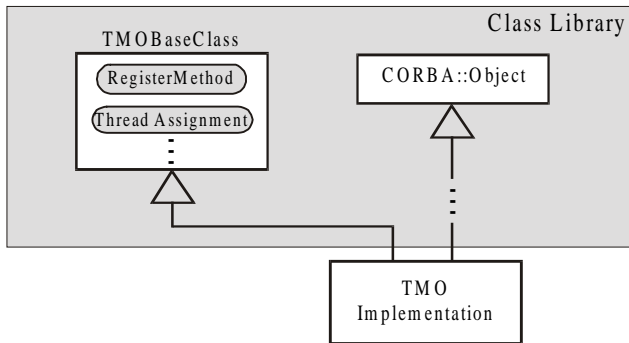


Figure 3: The Class Hierarchy of a TMO Class

## 2.3 TMOSM/OmniORB2/NT

TMOSM/OmniORB2/NT is a middleware built on top of the Windows NT operating system and OmniORB2; a freely-available CORBA-compliant ORB. It supports CORBA-compliant TMO structuring of distributed real-time applications. TMOSM/ORB provides a class library which includes (i) two base classes, TMOBaseClass and ODSSBaseClass, that simplify TMO based application development and (ii) an object execution engine that supports timely executions of SpM's and SvM's. Since the class library was discussed in Subsection 2.2, here we briefly discuss only the object execution engine. The main functions of the object execution engine are carried out by a special thread denoted here as TMOSM Thread (or TMOSMT). TMOSMT is an NT real-time thread [Ric97] that is periodically activated by a high-precision timer to manage the timely execution of the application. TMOSM/ORB also has additional internal threads providing support functions for TMOSMT. TMOSMT carries out the following important activities:

- Enforces timely execution of threads associated with the application SpM's or SvM's. TMOSMT identifies the thread to be executed at any given time (based on the timing attributes of corresponding registered TMO methods). When a candidate thread is selected for execution, other application/system threads are suspended to insure that the NT scheduler has no choice but to select the chosen candidate thread for execution.
- Timely activates TMOSM internal threads to carry out the support function needed for orderly execution of the TMO methods.
- Assigns specific time intervals during which all of the TMOSM activities (including the execution of application-level threads or TMOSM internal threads) are halted to allow Windows NT to perform its housekeeping activities.

## 3. Experience: Development of a real-time military application

In order to evaluate the effectiveness of TMOSM-based programming in simplifying design and implementation of real-world complex real-life applications and enhancing the application's robustness, we ported CAMIN (a Coordinated Anti-Missile Interceptor Network), which was originally implemented in the DREAM laboratory of University of California, Irvine [Kim97a, Kim97b], as a non CORBA-compliant network of TMO's running on a PC LAN, to the TMOSM/ORB environment.

### 3.1 CAMIN functionality

CAMIN is an anti-missile defense scenario in which a set of important targets (at least one ship) must be protected against hostile reentry vehicles. It consists of two major components; *the application environment simulator* and *the control computer network*:

- The application environment (also called theater) is a sky/land/sea segment of interest in which moving objects (such as the command-ship, and flying objects of both hostile and non-threatening types) appear in the thread randomly and move around. The application environment simulator creates the moving objects, moves them, and simulates operations and effects of intercepting actions taken by the control computer system's objects all in real-time with specified time granularities. The TMO structuring helps in accurate detailed modeling of the environment. More specifically, the environment can be simulated as accurately and precisely as the supporting hardware and software allow.
- The control computer network is responsible for defense computing functions, such as "tracking of flying items" and "discrimination of hostile reentry

vehicles from friendly or non-threatening flying items, such as ally fighters and birds”. If the control computer network distinguishes an object as threatening, it will order higher accuracy tracking of the object. If the object under surveillance turns out to be a hostile object, its interception process will then be initiated by the control computer network. In the current version of CAMIN, there is only one control center, but it can easily be extended to support several cooperative control centers. We plan to extend the CAMIN functionality to incorporate cooperative engagement capabilities among various control centers.

The initial top-level requirements given by the customer demand that (i) the hostile reentry vehicles be recognized within a given period of time and before reaching a specific range in which they may become practically dangerous, (ii) the sky-based airborne defense line (such as fighters) attempt to destroy the hostile items, and finally (iii) if the sky-based defense line is not successful in destroying hostile items, then the land-based defense line destroys the surviving hostile items.

### 3.2 TMO-based structuring of CAMIN

As discussed in the TMO literature [Kim94a, Kim94b, Kim97a, Kim97b], the TMO model can be used during both (i) the requirements specification, and high-level design steps and (ii) detailed design and implementation phases of the application development process. However, since our focus in this paper is to discuss the implementation model and related issues, we will not present the requirement specification and high-level design phase of the CAMIN development but concentrate on the detailed design and implementation phases.

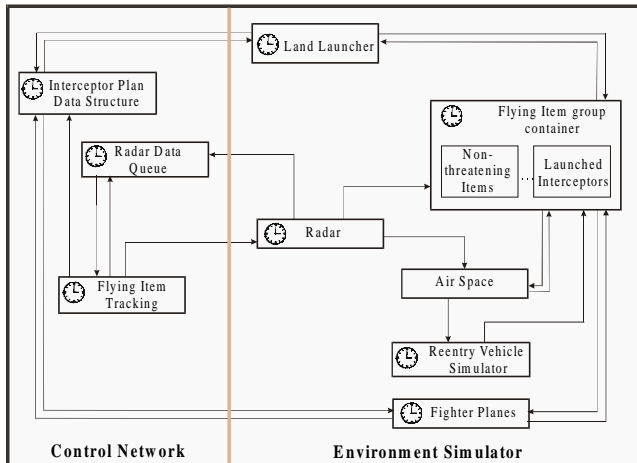


Figure 4: CAMIN as a Network of TMO's

Figure 4 presents the TMO structuring of CAMIN in the detailed design phase. The TMO's shown in the diagram are the smallest units of the CAMIN design and

are not partitioned further into smaller TMO's. An arrow from a TMO to another TMO indicates that the former may call one or more SvM's of the latter.

TMO's with one or more time-triggered methods (SpM's) are labeled with a clock. As shown in the figure, eight out of nine TMO's in CAMIN have at least one SpM. This illustrates the importance of the SpM construct, and thereby the practical benefit of TMO structuring, in design and implementation of complex real-time applications.

Timing attributes of SpM's and SvM's are identified at design time reflecting the timing characteristics of application-level activities. For example, the activation period of an SpM in RADAR-TMO that regularly scans the sky depends on the timing requirements (e.g., the maximum allowable time duration) for tracking and identifying a hostile reentry vehicle.

### 3.3 CAMIN as a network of CORBA-compliant TMO's

In order to implement CAMIN in the TMOSM/ORB environment, each TMO should be implemented as a CORBA-compliant TMO that was discussed in Section 2.3. To implement a TMO in a CORBA-compliant manner, it is mapped into a CORBA object that exposes its SvM's to the outside world as its interface. The SpM's and ODSS's are considered the internals of a TMO and will not be seen by the clients.

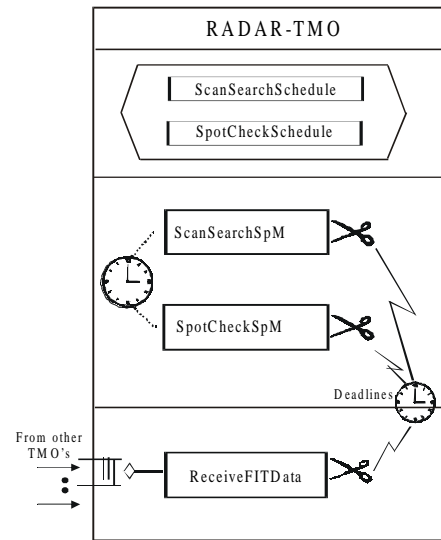


Figure 5: The Radar TMO

To further illustrate the process of compliant-TMO implementation, the implementation of RADAR-TMO of CAMIN will be briefly discussed in this subsection. The overall structure of RADAR-TMO is depicted in Figure 5. As shown in its graphical representation, the RADAR-TMO has the following elements:

- *Two SpM's*: ScanSearchSpM periodically scans various portions of the sky. Additionally, the radar can specifically check in a specified portion of the sky, provided that a client schedules such as spot-check. This spot-check is done by SpotCheckSpM.
- *One SvM*: ReceiveFITData is called by a client to pass the data about the flying objects to be spotted.
- *Two ODSS's*: ScanSearchSchedule and SpotCheckSchedule maintain information obtained by scan-search and spot-check respectively.

CORBA-compliant RADAR-TMO exposes its SvM to the outside world through its interface as shown in the following box:

```
interface RADAR-TMO {
    oneway void ReceiveFITdata(in
        SpotCheckPredictedType data_from_FIT);
};
```

The above interface definition illustrates the important point that a CORBA-compliant TMO is seen by clients as a regular CORBA object. This is an essential property of TMOSM based programming. On the other hand, the knowledge about SpM's and their timing attributes may help the client in identifying the timing characteristics of the services provided by the server object. This information, however, can be effectively exchanged among client developers and server developers in the detailed design phases.

The implementation of a CORBA-compliant TMO is a minor extension of the regular CORBA object implementation. To ease the implementation of CORBA-compliant TMO's, a class library is offered by TMOSM which encapsulates all generic functionality of TMO's: The TMOSM class library includes TMOBaseClass and ODSSBaseClass. The TMOBaseClass provides functions such as (i) registering the TMO methods with TMOSM, and (ii) Assigning threads to SpM's. It should be noted, however, that the programmer provides only the information about each method and TMOBaseClass will pass this information to TMOSM to be used for timing execution of the registered methods.

For illustrative reasons, the class specification of RADAR-TMO is shown in the following box. To make it more simple, only the relevant portions of the specification are shown here.

```
#include <TMO/TMOBaseClass.h>
#include "..\idl\RADAR.hh"
...
class RADARTMOClass :public TMOBaseClass,
                    public virtual _sk_RADAR_TMO
{
private:
    ScanSearchScheduleODSSClass
        ScanSearchScheduleODSS;
```

```
SpotCheckScheduleODSSClass
    SpotCheckScheduleODSS;

void ScanSearchSpM(void);
void SpotCheckSpM(void);

public:
void ReceiveFITdata(const SpotCheckPredictedType
    &data_from_FIT);

// Constructor contains registration info
RADARTMOClass(char* name,
    int NumSpMs, SpMParType *SpMParamArray,
    intNumSvMs, SvMParType *SvMParamArray);
~RADARTMOClass();
...
};
```

A few points about this illustrative example need to be briefly discussed. First, RADARTMOClass inherits from TMOBaseClass and sk\_RADAR\_TMO (a proxy class created by the idl compiler to act as a connection between the server object and its client). The way of relating an object implementation and its idl-created proxy is ORB-vendor dependant. In some ORB's, this is done using inheritance (such as the case with OmniORB), while in others it is achieved via object composition.

Another point is the definitions of SpM's and ODSS's of a TMO in the private section of the class definition hides them from clients. On the other hand, its RADARTMOClass's only SvM (i.e, ReceiveFITData) is defined as a public method.

The last, and perhaps the most important, point regards the structure of the CORBA-compliant TMO's. The construction of a CORBA-compliant TMO, as illustrated in the above code segment, must have the following parameters:

- NumSpMs: The number of SpM's defined in the class.
- SpMParamArray: An array, each element of which includes the information about a SpM. This information includes, among others, autonomous activation conditions and identification of the ODSS's used by the method.
- NumSvMs: The number of SvM's offered by the class.
- SvMParamArray: An array, each element of which includes the information about a SvM. This information includes, among others, the method execution deadline and identification of the ODSS's used by the method.

Using this information, the TMOBaseClass will be able to register SpM's and SvM's of the associated derived TMO class.

### 3.4 Evaluation of CORBA-compliant TMOSM-based real-time system development

Experience in porting CAMIN to the TMOSM/ORB environment validated our belief that the TMO structuring scheme plays an essential role in cost-effective and robust design and implementation of complex real-life real-time applications spanning from widely distributed real-time military applications to more-constrained applications, such as industrial automation. The TMO based structuring as well as programming in the TMOSM/ORB environment (CORBA-compliant companion of TMO structuring) offers the following specific benefits for the design and implementation of real-time applications:

- *TMO structuring offers a significant reduction in design and implementation time of complex real-time systems:* Since TMO structuring enforces a uniform hierarchical design and implementation of applications, it is much easier to decompose the application into several autonomous subsystems (or components) in an incremental fashion. This incremental decomposition of the application into TMO's is a natural way of conquering the complexity of large-scale applications. We have actively participated in the design and implementation of earlier versions of CAMIN using conventional process structuring schemes. Comparing the TMO-based implementation of CAMIN with efforts devoted to earlier versions of CAMIN, it is fair to say that the amount of effort spent for the design and implementation of the current version of CAMIN (which is much more complex compared to the earlier versions) is at least one order of magnitude less than the effort devoted to each of the earlier versions.
- *Timing requirements of various objects (and their services) are explicitly defined and effectively enforced in the TMOSM/ORB environment:* The two most important issues in developing complex real-time applications are (i) providing a natural way to specify time requirements of the application, and (ii) guaranteeing that the specified requirements will be enforced in a cost-effective manner. TMO structuring provides the SpM construct as a natural and effective technique for expressing time-triggered activities. It opens the way for design time validation of the application timing requirements. Moreover, the TMOSM/ORB execution engine is developed with the objective of enforcing the timing requirements of all SpM's and SvM's provided that the sufficient resources are available.
- *Maximum benefit from CORBA compliance:* CORBA standards provide a high-level clean abstraction for location-transparent language-neutral inter-object

communications. TMO's supported by TMOSM/ORB are seen by clients as regular CORBA objects. This means that a client can initiate and call the services provided by CORBA-compliant TMO's in the same way that the client can get services from regular CORBA objects. This tremendously simplifies the implementation of large-scale distributed systems.

- *Ease of programming (a minor extension of CORBA programming):* The TMO was created with the clear goal of making it easy for conventional object oriented business application software engineers to accommodate. In accordance with this important goal, the CORBA-compliant middleware support was designed such that (i) the application developers are relieved from the potentially error-prone details, such as thread management and synchronization issues and (ii) the application programmers are provided with a minimal set of system-calls through which the timing and data-sharing requirements of the application TMO objects can be specified. To accomplish this, TMOSM introduces a class library (which includes TMOBaseClass and ODSSBaseClass among others) encapsulating generic features of the TMO-based programming. As a result, TMO-based programming [Sho98c] is a minor extension of standard CORBA programming.
- *Resulting in more generic and maintainable code:* The above desirable characteristics collectively facilitate the realization of more maintainable and reusable software. With a very small effort, our development team was able to extend CAMIN by adding a sky-based defense line (i.e., adding fighter planes) to the application. This extension may require a massive restructuring of the process-structured CAMIN.
- *No change in CORBA features and minimal changes in the ORB implementation:* Current CORBA standards and the existing implementations do not explicitly address key characteristics of real-time systems, in particular, timing requirements. Thus real-time applications with strict timing requirements can not be best implemented in existing COTS CORBA-compliant ORB's. OMG has recognized this issue and created a special task group (Real-Time CORBA) to define real-time ORB standards [OMG96]. Our policy in using CORBA was (i) to not make changes to CORBA standards and the application interfaces and (ii) to extend the implementation such that the ORB runtime system becomes more responsive in various activities, such as time-bounded message transmission. To do so, as discussed earlier, we modified the OmniORB2 implementation to facilitate priority-based inter-node

communication. We believe this “no change to CORBA standards” approach makes sure that the TMO-structured programs are fully compliant with the standard CORBA.

#### 4. Conclusion

In this paper, we discussed the CORBA-compliant TMO based programming scheme supported by the TMOSM/ORB middleware that is built on top of the Windows NT operating system and a public domain CORBA-compliant ORB. We believe that the resulting CORBA-compliant TMO-based software development scheme significantly reduces the development and maintenance cost of large-scale real-time systems and also increases their overall robustness and dependability.

To evaluate the effectiveness of CORBA-compliant TMO structuring in simplifying real-time application development and enhancing the maintainability and reliability of the resulting applications, we ported a defense command-control application (named CAMIN) to the TMOSM/ORB environment.

One important finding from this experimental evaluation was that CORBA-compliant TMO-based application development provides a significant reduction in the design, development, and validation time of the application. It also significantly helps in developing more robust, maintainable, and extensible distributed real-time applications.

Another benefit of TMO-based software design that we also realized in our reported experience was traceability of timing characteristics of high-level actions (defined in requirement specification and detailed design phases) into the timing requirements of each individual TMO due to the hierarchical design of the application. This made the debugging of the CAMIN distributed program much easier, because all timing anomalies of the application can be more easily verified with the timing requirements of the two high-level TMO's (i.e., environment and control computer network).

#### 5. References

[Boo94] Booch, G., *Object Oriented Analysis and Design with Applications*, Redwood City, CA: Benjamin/Cummings Publishing, 2nd ed., 1994.  
[Kim94a] Kim, K.H., et al, “Distinguishing Features and Potential Roles of RTO.k Object Model”, *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems*, Feb. 1994, Laguna Beach, CA, pp. 59-68.  
[Kim94b] Kim, K. H., and Kopetz, H., “A Real-Time Object Model RTO.k and an Experimental Investigation

of Its Potentials”, *Proc. 1994 IEEE CS Computer Software Applications Conference*, Nov. 1994, Taipei, pp. 392-402.

[Kim97a] Kim, K. H., "Toward New-Generation Object-Oriented Real-Time Software And System Engineering", Invited paper, *SERI Journal*, Vol. 1, No. 1, Jan. 1997, pp. 1-23.

[Kim97b] Kim, K. H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No. 8, August 1997, pp. 62-70.

[Kim97c] Kim, J. G., et al, “Multimedia Service Object Modeling”, *Proc. 1997 IEEE CS Workshop on Object-oriented Real-time Dependable Systems*, Feb. 1997, Newport Beach, CA, pp. 347-354.

[Lo98] Lo, S. L., *The OmniORB2 Version 2.4: User Guide*, Olivetti & Oracle Research Laboratory, Jan. 1998.

[Ngu97] Nguyen, C. M., and Kim, K. H., “Toward Optimal Assignment of Human Functions in Complex Defense Systems Via Uniform Object Modeling and Real-Time Simulation”, *Proc. 1997 IEEE CS Workshop on Object-oriented Real-time Dependable Systems*, Feb. 1997, Newport Beach, CA, pp. 332-338.

[OMG95] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, 1995.

[OMG96] OMG Real-Time Interest Group, “Real-Time CORBA Issues 1.0”, *Working Document, OMG*, 1996.

[Ric97] Richter J., *Advanced Windows*, 3<sup>rd</sup> Edition, Microsoft Press, 1997.

[Sho97] Shokri, E., et al, “ROAFTS: A CORBA-Based Real-Time Fault Tolerance Support Middleware”, *Proc. IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, Dec. 1997, San Francisco, CA, pp. 262-268.

[Sho98a] 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. 1998 International Symposium on Object-oriented Distributed Computing (ISORC'98)*, April 1998, Kyoto, Japan, pp. 12-20.

[Sho98b] Shokri, E., Crane, P., Kim, K. H., and Subbaraman, C., “Architecture of ROAFTS/Solaris: A Solaris-Based Middleware for Real-Time Object-oriented Adaptive Fault Tolerance Support”, *Proc. 1998 IEEE CS Computer Software Applications Conference*, August 1998, Vienna.

[Sho98c] Shokri, E., and Zhang, B., “TMOSM/OmniORB/NT Programming Guide: Version 1.0”, *SoHaR Inc. Internal Technical Document*, Oct. 1998.

PROCEEDINGS  
1999 IEEE SYMPOSIUM ON  
APPLICATION-SPECIFIC SYSTEMS AND  
SOFTWARE ENGINEERING & TECHNOLOGY  
ASSET'99

---

MARCH 24-27, 1999

CLARION HOTEL

RICHARDSON, TEXAS

**Co-Sponsored by**

IEEE Computer Society  
and

Center for Application-Specific Systems and Software Engineering (CASSE)  
at UT-Dallas



Los Alamitos, California

Washington • Brussels • Tokyo

---