

Distributed Object-Oriented Real-Time Simulation of Ground Transportation Networks with the TMO Structuring Scheme

K. H. (Kane) Kim, Juqiang Liu, Masaki Ishida

Dept. of Electrical & Computer Engineering
University of California
Irvine, California, U.S.A.

{kane, jqliu, masaki} @ece.uci.edu

and Inho Kim

Dept. of Chemical Engineering,
Chungnam University
Daeduk, Korea

(A former visiting researcher at UCI)

Abstract: The TMO (Time-triggered Message-Triggered Object) structuring scheme is aimed at facilitating real-time (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. In the course of developing a TMO-structured RT system engineering methodology, an attractively simple approach to parallel and distributed RT simulation has been produced. Also, we recently implemented a TMO execution engine based on the Windows NT platform and named it TMOSM/NT. As an effort for validation of both the execution engine TMOSM/NT and the TMO-structured RT simulation approach, a freeway automobile traffic simulator named distributed object-oriented freeway simulator (DOFS), has been constructed. In this paper, the design techniques and tools applied in development of DOFS are discussed.

1. Introduction

The TMO (Time-triggered Message-Triggered Object) structuring scheme has been established to remove the limitation of conventional object structuring techniques in developing real-time (RT) applications [Kim94, Kim97]. 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. 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 the course of developing a TMO-structured RT system engineering methodology, a new approach to RT simulation which is conceptually simple and easy to use but widely applicable, has also been produced [Kim94c, Kim96b]. It is an attractively simple approach to parallel and distributed RT simulation. Here the term *RT simulation* refers to an advanced mode of simulation in which the simulation objects are designed to show the timing behavior that is the same or similar to the timing behavior of the simulation targets.

Recently a model of a TMO execution engine which is centered around a middleware running on a typical commercial operating system kernel has been developed and a prototype implementation based on the Windows NT platform, named TMOSM/NT, has been produced. As an effort for validation of both the execution engine TMOSM/NT and the TMO-structured RT simulation approach, a freeway automobile traffic simulator named *distributed object-oriented freeway simulator* (DOFS), has been constructed.

In this paper, the design techniques and tools applied in development of DOFS are discussed. DOFS is intended to support serious studies of advanced freeway management systems by providing *high-resolution high-accuracy easily expandable* freeway simulation. DOFS is also meant to be a foundation for future expansion into a high-fidelity regional traffic simulator which simulates not only freeways but also stop-and-go surface streets.

The paper starts in Section 2 with a brief overview of the TMO structuring scheme and the TMOSM architecture. Then the RT simulation approach facilitated by the TMO scheme is reviewed. In Section 3, the design of DOFS is discussed. Section 4 discusses the advantages of the TMO-structured distributed / parallel RT simulation approach that have been exhibited in the DOFS development experiment. The paper concludes in Section 5.

2. Backgrounds

2.1 An overview of the TMO structuring scheme

The *Time-triggered Message-triggered Object* (TMO) structuring scheme was established in early 1990's [Kim94, Kim97b] with a concrete syntactic structure and execution semantics for economical reliable design and implementation of RT systems. The basic TMO structure is depicted in Figure 1.

TMO is a syntactically minor and semantically powerful extension of the conventional object(s). Significant extensions are summarized below and the second and third are the most unique extensions.

(a) *Distributed computing component:*

The TMO is a distributed computing component and thus TMO's distributed over multiple nodes may interact

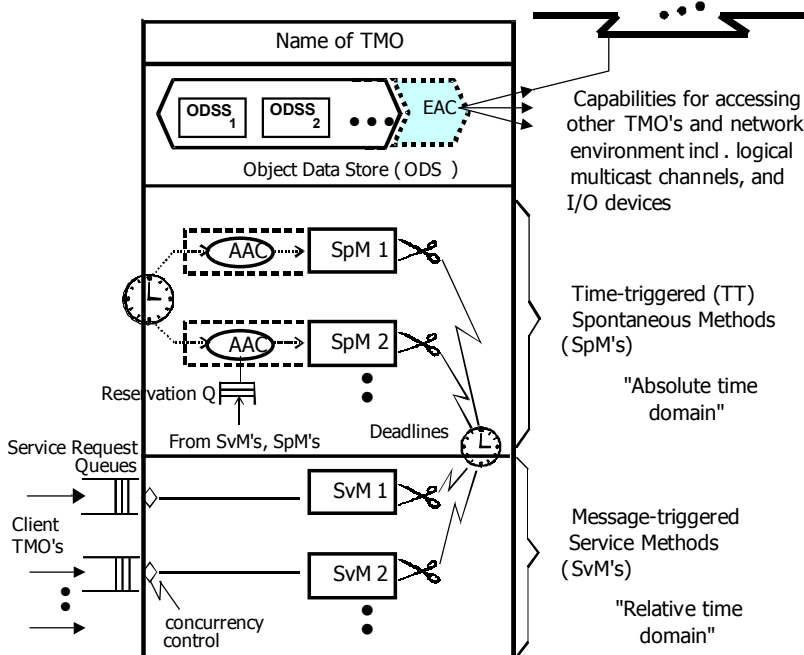


Figure 1. The basic structure of TMO (Adapted from [Kim97])

via remote method calls. To maximize the concurrency in execution of client methods in one node and server methods in the same node or different nodes, client methods are allowed to make non-blocking types of service requests to server methods.

(b) *Clear separation between two types of methods:*

The TMO may contain two types of methods, *time-triggered (TT-) methods* (also called the *spontaneous methods* or *SpM's*), which are clearly separated from the conventional *service methods (SvM's)*. The SpM executions are triggered upon reaching of the RT clock at specific values determined at the design time whereas the SvM executions are triggered by service request messages from clients. Moreover, actions to be taken at real times *which can be determined at the design time* can appear only in SpM's.

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

This rule prevents potential conflicts between SpM's and SvM's and reduces the designer's efforts in guaranteeing timely service capabilities of TMO's. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place*. An SvM is allowed to execute only if no SpM that accesses the same object data store segments (ODSS's) 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.

(d) *Guaranteed completion time and deadline:*

As in other RT object models, the TMO incorporates

deadlines and it does in the most general form. Basically, for output actions and method completions of a TMO, the designer guarantees and advertises execution time-windows bounded by start times and completion times.

Extensions (b) and (c) are unique to the TMO structure in comparison with other proposed object extensions [Att91, Ish90, Tak92]. Triggering times for SpM's must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the *autonomous activation condition (AAC)* section. An example of an AAC is

"for t = from 10am to 10:50am every 30min start-during (t, t+5min) finish-by t+10min"

which has the same effect as

{ "start-during (10am, 10:05am) finish-by 10:10am",

"start-during (10:30am, 10:35am) finish-by 10:40am" }

A provision is also made for making the AAC section of an SpM contain only *candidate* triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM within the same TMO object requests future executions of a specific SpM. The AAC specifying candidate triggering times rather than actual triggering times starts with a declaration "if-demanded".

An underlying design philosophy of the TMO scheme is that an RT computer system will always take the form of a network of TMO's. TMO's interact via calls by client objects for service methods in server objects. The caller may be an SpM or an SvM in the client object. To maximize concurrency in the execution of client and server methods, client methods are allowed to make non-blocking (sometimes called asynchronous) types of service requests to SvM's.

The designer of each TMO provides a guarantee of timely service capabilities of the object. He/she does so by indicating the *guaranteed execution time-window for every output* produced by each SvM as well as by each SpM executed on requests from the SvM and the *guaranteed completion time* for the SvM in the specification of the SvM. Such specification of each SvM is advertised to the designers of potential client objects. Before determining the time-window specification, the server object designer must convince himself/herself that with the *object execution engine* (hardware plus operating system) available, the server object can be implemented

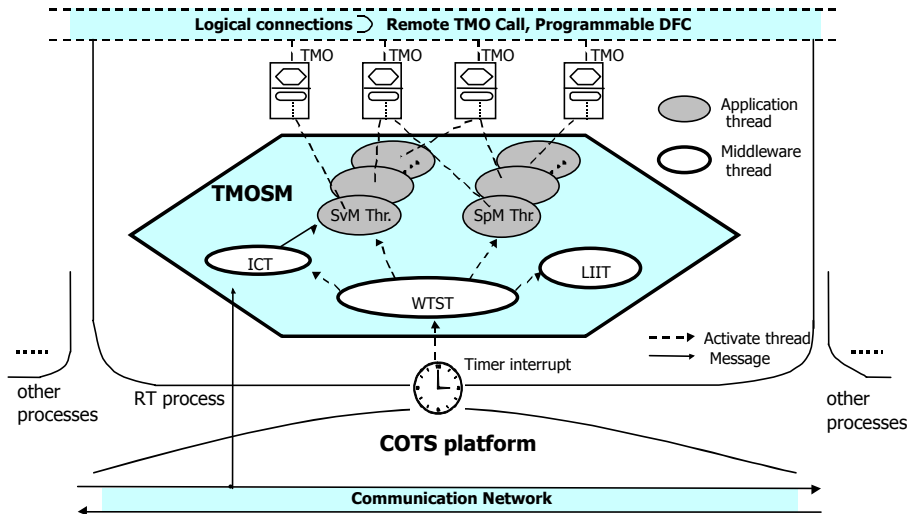


Figure 2. The basic internal thread structure of TMOSM (adapted from [Kim99])

to always execute the SvM such that the output action is performed within the time-window. The BCC contributes to major reduction of these burdens imposed on the designer.

2.2 TMO Support Middleware (TMOSM) architecture

TMOSM is a TMO execution support middleware model which can be easily adapted to most COTS platforms [Kim99]. The internal thread structure of TMOSM is shown in Figure 2.

TMOSM consists two types of threads, the *application thread* and the *middleware thread* (also called system thread). TMOSM assigns one application thread to each SpM or SvM of an application TMO. Middleware threads are periodic threads (periodically activated to run for a time-slice), each being responsible for a major part of the functions of TMOSM. The set of middleware threads is fixed at the TMOSM start time. The following four middleware threads handle the core functions:

- (1) WTST (Watchdog Timer & Scheduler Thread): This periodic thread manages the scheduling / activation of all other threads in TMOSM and checks if there are deadline violations. If any violation is found, it provides an exception signal to the user.
- (2) ICT (Incoming Communication Thread): This thread manages the distribution of messages coming through the communication network to their destination threads.
- (3) LIIT (Local I/O Interface Thread): This thread manages local I/O activities such as serial character I/O and Disk I/O.

- 4) VMST (Virtual Main System Thread): Every time-slice not used by the above three middleware threads is conceptually given to this virtual thread which merely represents all application threads. The actual time-slice allocations are taken by WTST which executes the application scheduler function and every time-slice conceptually belonging to VMST is allocated to a fairly selected application thread.

The first prototype implementation of TMOSM, TMOSM/NT, has been developed recently in the authors' laboratory (DREAM lab of UCI). From the validation tests conducted, we have found that TMOSM/NT can accurately enact the time-window

for activating a method as small as 10ms and the method completion deadline as short as 20ms for more than 99.9% of the time [Kim99].

To facilitate economic TMO-structured programming, a user-friendly API function library, named the *TMO Support Library* (TMOSL), has been created. It consists of a collection of C++ classes and functions for RT application programmers, including Basic TMO class, RT clock service, and RT I/O functions.

2.3 Real-time simulation facilitated by TMO

In an RT simulator, the *simulator clock* must “tick” at a steady rate. Each tick of the simulator clock is commenced and administered by referencing an RT clock in the *simulation execution engine* (a computer running the simulation program). The ticking rate of the simulator clock in an RT simulator must be chosen such that during any ticking interval, all (real-time) events which should be simulated during that interval may be treated as being “contemporary”. In other words, the microscopic order in which events are simulated during a ticking interval should be immaterial as far as simulation results are concerned. This means that all events may be treated as ones occurring at the end of the ticking interval (i.e., right before the next ticking of the simulator clock). This is a fundamental requirement, which may be called *the simulator clock atomicity requirement* [Kim96]. All computational activities taking place during a ticking interval of the simulator clock may be viewed as one *simulation-step*.

For example, consider the case of simulating automobiles moving on a freeway. Since the moving pattern of each automobile is affected by the moving automobiles in the neighborhood area, the selection of the

ticking interval of the simulator clock is an important matter. If the ticking interval is chosen to be 5 seconds, then the state of every automobile will be updated every 5 seconds. However, if a typical automobile can change its lane in one second and a lane change by an automobile can have big impacts on subsequent behaviors of some other automobiles, updating the states of all automobiles every 5 seconds means a very low-fidelity low-accuracy simulation. Especially, if automobile collisions are to be dealt with in this simulation, it is possible that a collision occurring at a certain time can lead to the same simulation result as a collision occurring 4.9 seconds later does. This is due to the simulator clock atomicity requirement. That is, all collisions occurring during a ticking interval may be treated as ones occurring at the end of the ticking interval.

A freeway-segment can be represented and simulated by the TMO in Figure 3.

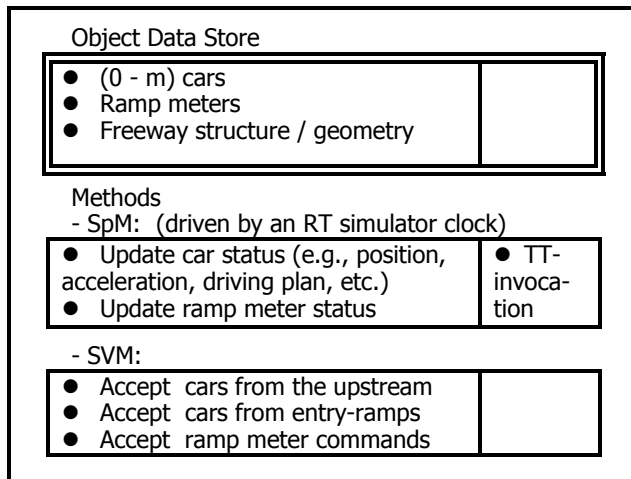


Figure 3. A TMO-structured simulation model for a freeway-segment

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, when executed, updates a variable-set in the ODS representing the state of some physical item (i.e., car, ramp meter) to reflect the current state of the physical 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 RT clock in the execution engine. The activation frequency of the TT-method may be viewed as *the ticking rate of the physical item simulator clock*. Each execution of a TT-method must be completed within one ticking interval of the physical item simulator clock. Therefore, TT-methods are the mechanisms for approximately simulating continuous state changes that occur naturally in the items in the environment.

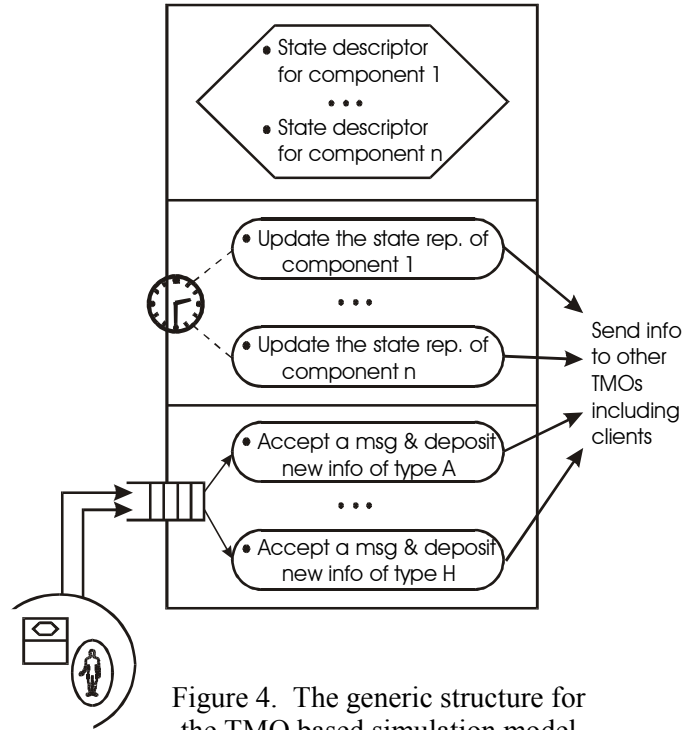


Figure 4. The generic structure for the TMO based simulation model

The natural parallelism that exists among the physical items in the environment is precisely represented by use of multiple TT-methods which may be activated simultaneously. In general, the accuracy of a TMO-structured simulation of the environment is a direct function of the activation frequencies of TT-methods (which are equivalent to the ticking rates of the physical item simulator clocks).

Figure 4 depicts the TMO-structured RT simulation approach in a generic form. An abstract TMO can be expanded into a network of more detailed TMO's. This can be done by dividing the ODS of the abstract TMO into multiple ODS parts and constructing an independent TMO around each ODS part.

The simplicity of the structure depicted in Figures 3 and 4, the possibility of systematic expansion of a TMO into a TMO network, and both the global time base support and the abstract programming support from the networked cooperating TMO execution engines in RT distributed computing systems, represent a significant potential for improving the economy and efficiency of distributed / parallel RT simulation over the state of the art.

3. Design and implementation of DOFS

The design goal of DOFS is to create a facility which can be tuned with small efforts to simulate with a high degree of accuracy and precision the behaviors of cars on freeways in real time or scaled real time. Easy

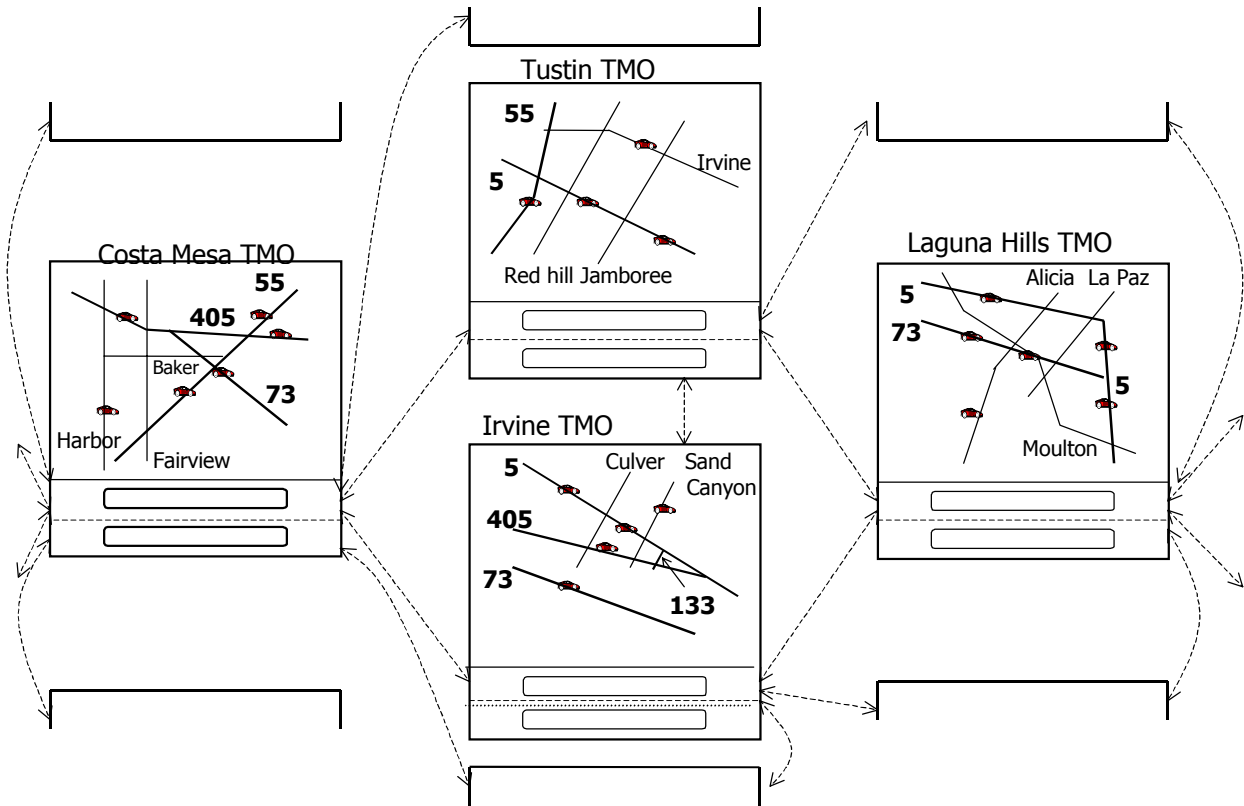


Figure 5. Partitioning of an area for distributed simulation

exploitation of distributed / parallel processing was another design goal and this is important with respect to making the simulator scalable. As a distributed computing component technology, the TMO scheme allows the designer / user to exploit distributed / parallel processing with minimal efforts. In the case of the DOFS, the entire simulator takes the form of a TMO network in which each TMO simulates one part of the freeway. See Figure 5 which illustrates this geographical partitioning and the coverage of partitions by different simulator TMO's.

One TMO updates the status of all the cars in the region it simulates. It can also communicate with other TMO's to send/receive cars to/from them as depicted in Figure 5.

3.1 Garage model

In the specific DOFS configuration, DOFS-405NB, a segment of freeway 405NB (North Bound) in Orange County was chosen as the simulation target. Other freeways or local streets which are connected to freeway 405NB are viewed conceptually as "garages". Therefore the freeways which send cars to freeway 405NB are modeled as an *upstream garage*; the freeways which receive cars from freeway 405NB are modeled as a *downstream garage*; the local streets which connect with freeway 405NB are modeled as a *local street garage*. This simulation model is depicted in Figure 6.

The high-level functions of this simulation model can be summarized as follows:

- Generate cars in the upstream and local street garages periodically and send them to the freeway.
- Update the status of all the cars in the freeway periodically. The behavior of an individual car depends on the car type, the driver type, its destinations, and its neighbors' behavior, etc.
- There are ramp meters in the entry-ramps to control the rate of cars going into the freeway. By changing the ramp meter rates we can influence the congestion condition in the freeway.
- When a car goes off the freeway-segment being simulated in detail and enters the freeway-segment in the down stream, the freeway TMO should send the car to the downstream garage TMO. Or if a car goes off through an exit-ramp according to its schedule, then the freeway TMO should send the car to the local (street) garage TMO.
- The local garage may accept cars exiting from the freeway-segment at a rate dictated by the traffic condition in the relevant local streets.

3.2 Top-down design

Following the top-down systematic expansion methodology facilitated by the TMO structuring, the

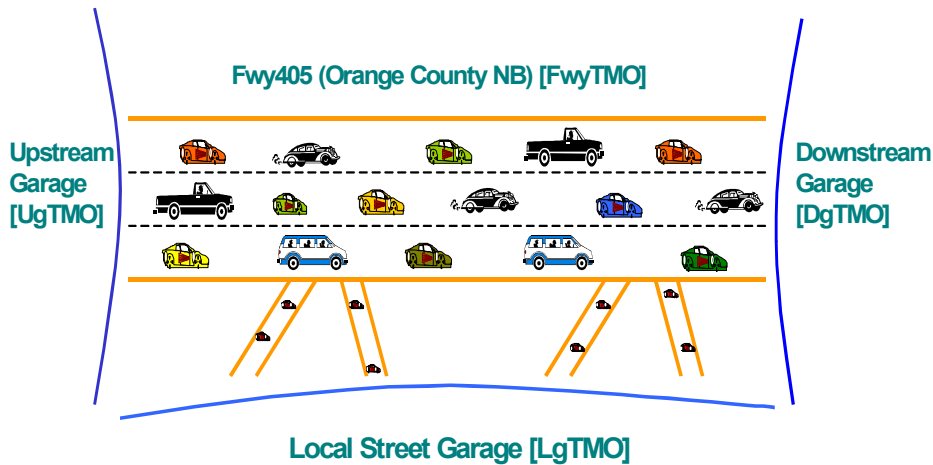


Figure 6. Garage models

entire environment to be simulated is initially represented by one TMO. One driver for this expansion is to partition the freeway region covered by one TMO into several sub-regions each of which will be covered by an independent TMO after the expansion. Another reasonable partitioning is to separate the freeway structure from the control subsystem that controls the actuators such as ramp meters, electronic message signs, etc.

A simulator for the small freeway environment depicted in Figure 6 may be built as a single TMO. Or it can be built as a network of four TMO's: one for the freeway-segment of main interest, and the other three for the three different types of garages. If the TMO covering the freeway-segment of main interest becomes too large, it can be partitioned further into multiple TMO's, each covering a smaller segment of the freeway.

In addition, the control system which controls actuators may be built as a TMO or TMO network. In the

case of controlling entry-ramp meters, the controller TMO (or TMO network) may calculate and control the meter rates according to the car density and speeds in the relevant sections of the freeway. Such a TMO-structured control system can be tested with the TMO-structured freeway simulator before its deployment in the field. Then the controller TMO (network) will receive sensor data such as current car density and speeds or similarly relevant data from the simulator TMO network.

3.3 Synchronization among simulator TMO's

Since information must be communicated among the simulator TMO's in timely manners, synchronization among the simulator TMO's is a fundamental issue. Conventional distributed / parallel simulation approaches involve message communication among the simulator nodes for the purpose of keeping them synchronized. Under those approaches, the number of messages increases rapidly when the number of interacting simulator nodes increases. Under the so-called optimistic approaches which allow short-term synchronization-free executions, synchronization violations may be detected after such executions, which then incur expensive rollbacks.

Under the TMO scheme, no synchronization message is needed. This is because each simulation step is time-triggered in every simulator TMO. Therefore, once the triggering time of each step in the distributed simulation system is chosen to be the same in every simulator TMO and the *simulator clock tick* (i.e., the triggering interval of each SpM in simulator TMO's) is chosen to be

sufficiently long to cover the execution of a simulation step of every participating TMO, the simulator TMO's can proceed simultaneously without exchanging synchronization messages.

Figure 7 depicts the synchronization aspects of the TMO-structured distributed RT simulation in more detail. The interaction between the *freeway TMO* and the *downstream garage TMO* is shown.

The downstream garage TMO sends the current status of the cars in its jurisdiction to the freeway TMO. This is because the

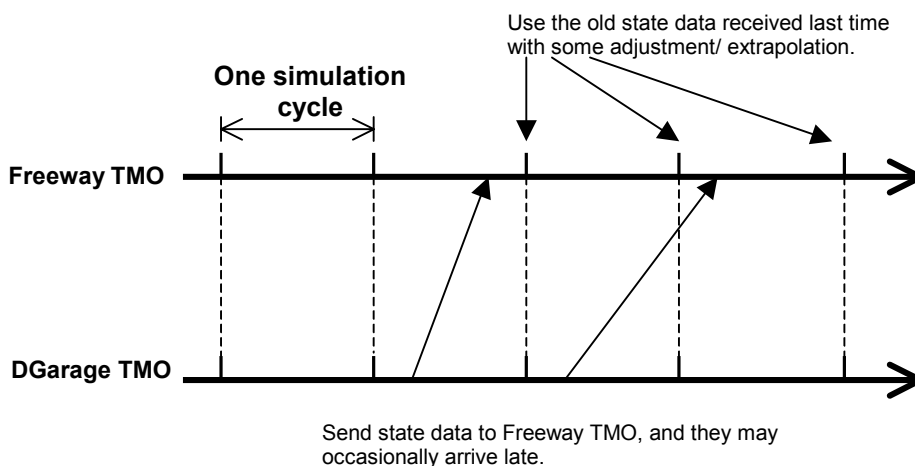


Figure 7. Synchronization between simulator TMO's

congestion conditions in the downstream garage affects the behavior (e.g., speed) of the cars on the freeway near the boundary between the freeway and downstream garage. The scheme in Figure 7 allows occasional late arrival of messages at the freeway TMO by incorporating the state extrapolation logic. In other words, if the status message from the downstream garage TMO is not available when the new simulation step begins, then the freeway TMO applies an extrapolation logic to the status messages received in the most recent rounds and obtains a reasonable estimate of the current status. Of course, this will work only up to a certain point of message delay.

3.4 Implemented TMO's

The logic incorporated in our implementation of DOFS is now briefly sketched.

The *upstream garage TMO* consists of two SpM's. One SpM generates cars using a set of car generation functions. The other SpM sends cars to the freeway by calling an SvM of the freeway TMO.

The *local street garage TMO* consists of two SpM's and one SvM. The functions of the two SpM's are similar to those of the SpM's in the upstream garage TMO, i.e., generate cars and send them to the freeway TMO. The SvM in the local garage TMO is called by the freeway TMO to simulate the transition of the cars exiting the freeway into local streets. The SvM calculates the next *absorption time*, i.e., next time when the local street can accept a car coming off the freeway via the exit ramp, using a set of car-absorption functions. The SvM sends the next absorption time to the freeway TMO.

The *downstream garage TMO* consists of one SpM and one SvM. The SpM updates the status of the cars in the downstream garage and sends the current status to the freeway TMO. As mentioned earlier, this is because the congestion condition in the downstream garage affects the behavior (e.g., speed) of the cars in the freeway near the downstream garage. The SvM of the downstream garage TMO is called by the freeway TMO to receive the cars moving off the freeway into the downstream garage. Therefore, there are two-way communications occurring between the downstream garage TMO and the freeway TMO.

The freeway TMO consists of one SpM and five SvM's. Some details of the design of the freeway TMO are in Figure 8. The SpM updates the status of all the cars in the freeway and ramps periodically and sends cars to the local street garage or the downstream garage if necessary. The SpM classifies the cars into the following eight groups and simulates different behaviors for each group:

Case 1: The cars whose birth-times have not arrived;

Case 2: Steadily moving cars;

Case 2.1: Steadily moving cars on the freeway;

Case 2.2: Steadily moving cars on a ramp;

Case 3: The cars going into the downstream garage;

Case 4: The cars in the fading lane;

Case 5: The cars going into the local street garage from an exit-ramp;

Case 6: The cars on entry-ramps and in the near front of one ramp-meter;

Case 7: The cars on the freeway with their destination exit-ramps approaching;

Case 8: The cars in the end of entry-ramps and ready to enter freeway;

First, the logic used in the SpM in updating the car status includes two parts: one is related to lane-changing decision and execution, while the other is related to speed and position changing. In the lane-changing decision and execution logic, a random number generator is used and parameters such as the current speed, position and destination of the subject car, the speeds and positions of the following and leading cars in the same lane and adjacent lanes, are reflected. For speed and position changing, parameters reflected include the speed and position of the subject car and the speeds and positions of the following and leading cars in the same lane.

The nature of the five SvM's is self-explanatory.

SvM1 is called by the upstream garage TMO to simulate the transition of newly-born cars from the upstream garage into the freeway segment; SvM2 is called by the downstream garage TMO to receive the cars' status from the downstream garage; SvM3 is called by the local street garage TMO to receive new-born cars from the local street garage; SvM4 is called by the local street garage TMO to update the next absorption time of the local street garage; SvM5 is called by the controller TMO to update the current ramp-meter rate status.

3.5 Implementation status

The current version of DOFS runs on Windows NT platforms equipped with the middleware TMOSM/NT. It is a microscopic freeway traffic simulator in which the behavior of each individual car is tracked in real time. The most basic configuration consists of five TMO's running on two NT nodes and one graphic node. A simulator of a section of Freeway 405NB (about 25 miles) has been established and it runs with the simulator clock tick of 0.5 second. The graphic display node uses non-RT objects. It was developed using MFC (Microsoft Foundation Class) Tools and displays the RT status of the simulator. It shows the freeway situation in multiple levels of details, and users can zoom in /out between different levels. There are two basic display modes, the *car density display mode* and the *individual car tracing mode*. In high-level displays, the density information is shown whereas in detail-level displays, individual cars are shown. The graphic display node can interact with the simulator nodes and request the latter to send only the

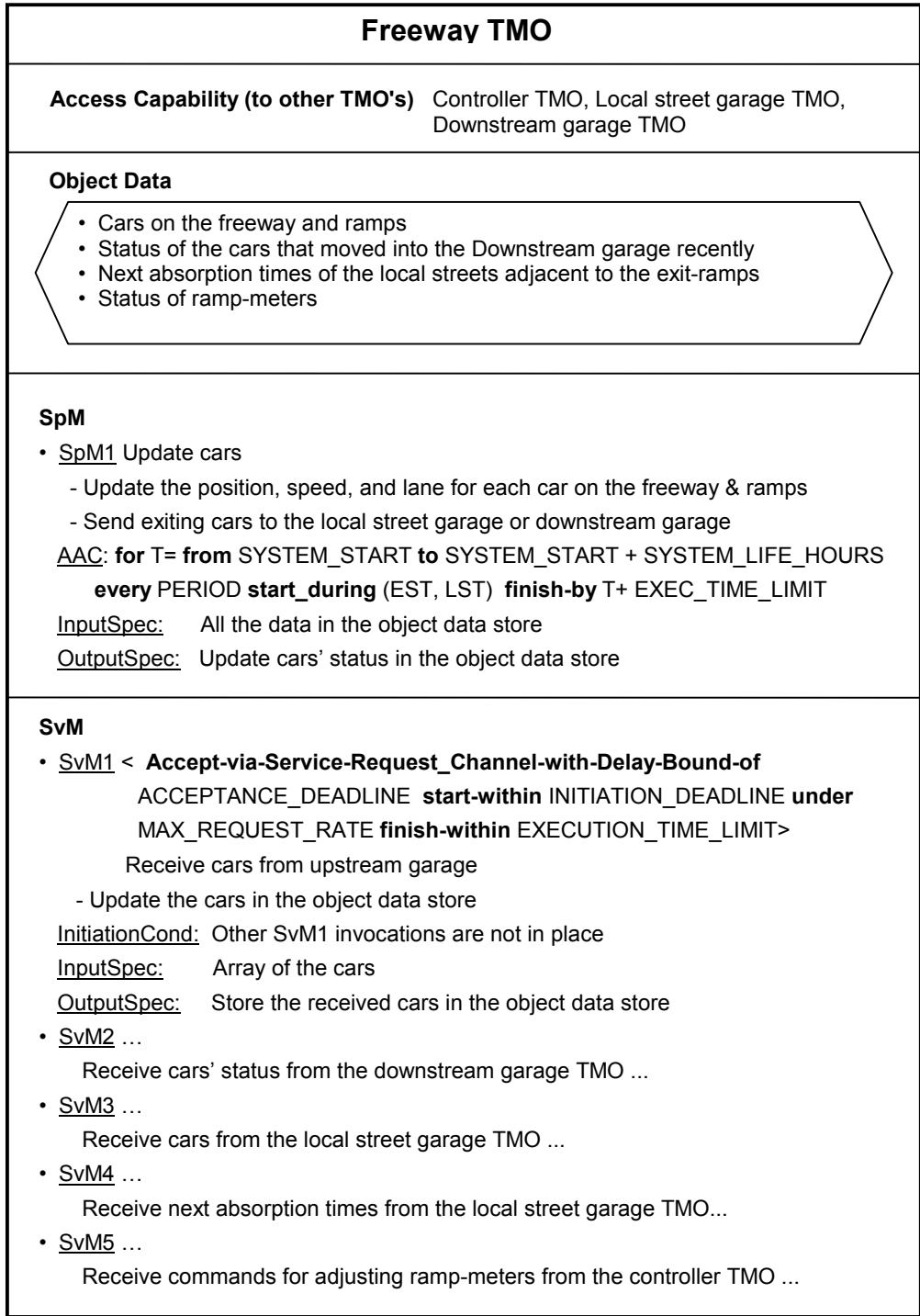


Figure 8. Freeway TMO

information about the cars in a selected displayable region of the freeway.

During the development of this freeway simulator and other TMO-structured applications, we observed that the programming and debugging efforts were significantly

less than the efforts required during our earlier development of similar but simpler applications using the conventional object-oriented design or other methods. We thus believe firmly that the TMO structuring scheme supported by the tools such as TMOSM / NT bring in major improvement in the RT system design and implementation efficiency.

4. Advantage of the TMO structuring and TMOSM architecture

During our development of DOFS (and other TMO-structured RT distributed applications), we observed quite a few advantages of the TMO structuring scheme supported by TMOSM/NT. Some major ones are as follows:

Uniform structuring: The same notation, structuring styles, and refinement procedures are used from the requirement specification to the detailed implementation of each TMO. Therefore, the relationship between the requirement specifications and designs can be easily recognized.

Highly predictable timing performance: The timing characteristics of the subject RT application are explicitly specified during the design time in each TMO specification. The TMO execution engine, TMOSM, tries to meet the timing

specifications of each SpM or SvM accurately. Executions of time-consuming temporally unpredictable I/O operations, such as screen output, keyboard input, file I/O, etc., are handled by one middleware thread, LIIT (Local I/O Interface Thread). TMO method execution

threads are not burdened with these tasks. Thus the highly predictable timing behavior of SpM's or SvM's in execution is not impaired.

Easy to implement and debug: With conventional methods, the most difficult parts of distributed RT application development are the programming and debugging of the timing characteristics and the concurrency control. Using a user-friendly API, TMOSL, designers can leave much of the detailed timing and concurrency control tasks to the TMOSM.

Easy to modify and expand: Each TMO in the TMO network is highly autonomous. Much of the time-sensitive computations are done by the SpM's which have no execution dependency and message communication among themselves. Moving a TMO from one node to another node can be done with small efforts. Scaling up or down in time dimension is easy. To accommodate a widely varying degree of the fidelity of the required simulation or to adapt to a wide range of computing power of different simulator execution engines, we often need to scale up or down the entire simulation system in the time dimension. Under conventional approaches, multiple modifications might be needed, but in the case of TMO-structured RT simulation, all that is needed is to change the scale of the RT clock. Let STR represent the ratio of speed(simulator-clock) to speed(real-time-clock). Then, when the SpM in a simulator TMO is specified to run at the frequency of one per second and STR is set to 1/4, it will actually run at the speed of one per 4 seconds. If the hardware part of the TMO execution engine changes, e.g., to a faster CPU, then only STR needs to be changed and other modifications are not necessary.

Efficient distributed and parallel processing in heavy-load simulations as discussed in Section 3.3.

5. Conclusion

The DOFS development was one of several experiments that we conducted with the goal of validating the potential of the TMO structuring scheme supported by the recently implemented TMO execution engine, TMOSM/NT. The DOFS experiment has become another confirmation of the potential of the TMO scheme for yielding significant reduction in the development cost and significant increase in the dependability of the RT applications constructed. In addition, DOFS is intended to be a tool for serious researchers in the field of advanced transportation management systems. We plan to use DOFS to study the traffic behavior in our frequently congested county and obtain results that are of good value to the traffic management system operators. However, to fully realize the potential of the TMO structuring scheme, much further analytical and experimental research including development of timing analysis tools, is needed.

Acknowledgments: 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, in part by the California State Department of Transportation under the ATMS Testbed Research Program of UCI ITS, in part by Postech, and in part by Hitachi, Ltd.

Reference

- [Dah72] Dahl, O.J., "Hierarchical Program Structuring", in Dahl, Dijkstra, & Hoare eds., '*Structured Programming*', Acad. Press, NY, 1972.
- [Ell93] Ellenberger, R., Ling, R., Buscher, D., Uhde-Lacovara, J., and Shuler, R., "Automatic Generation of Real-Time Ada Simulations for Space Station Freedom", *Simulation*, Nov.1993, pp.337-345.
- [Fuj90] Fujimoto, R.M., "Parallel Discrete Event Simulation", *Communications of the ACM*, Vol. 33, No. 10, Oct. 1990, pp.30-53.
- [Guy94] Guyse, C., Buscher, D., and Ellenberger, R., "Real-time Environment and Vehicle Dynamics Simulations for Space Station Freedom Integrated Test and Verification Environment", *Simulation*, Apr.1994, pp.230-239.
- [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. 94, Dana Point, pp. 36-45.
- [Kim96] Kim, K.H., et al, "The DREAM Library Support for PCD and RTO.k programming in C++", *Proc. 1996 IEEE CS Wworkshop on Object-oriented Real-Time Dependable Systems (WORDS)*, February 1996, Laguna Beach, CA, pp. 59-68.
- [Kim97] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No.8, August 1997, pp. 62-70.
- [Kim98] Kim, K.H., Subbaraman, C., "Principles of Constructing a Timeliness-Guaranteed Kernel and Time-triggered Message-triggered Object Support Mechanisms", *Proc. 1st IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC '98)*, Kyoto, Japan, April, 98, pp.80-89.
- [Kim99] Kim, K.H. Ishida, Masaki, Liu, Juqiang, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. 2nd IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC '99)*, St. Malo, France, May, 1999, pp.54-63.
- [Mis86] Misra, J., "Distributed Discrete-Event Simulation", *ACM Computing Surveys*, Vol. 18, No. 1, March 1986, pp.39-65.

Proceedings

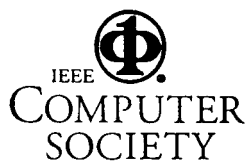
**THE TWENTY-THIRD ANNUAL
INTERNATIONAL COMPUTER SOFTWARE
AND APPLICATIONS CONFERENCE**

OCTOBER 27 – 29, 1999

Phoenix, Arizona

SPONSORED BY

IEEE Computer Society



Los Alamitos, California

Washington • Brussels • Tokyo
