Proc. ISORC 2001 (4th IEEE CS Int'l Symp. on OO Real-time distributed Computing), Magdeburg, Germany, May 2001, pp. 41-50.

The Distributed Time-Triggered Simulation Scheme Facilitated by TMO Programming

K. H. (Kane) Kim
University of California
Irvine, CA
khkim@uci.edu

Raymond Paul
US Dept of Defense
Pentagon, VA
Raymond.Paul@osd.mil

Abstract: Real-time simulation is an advanced mode of simulation in which the simulator components are designed to show essentially the same timing behavior that the simulation targets do. Distributed real-time simulation is a field in its infancy but its practice is under increasing demands. In recent years the author and his collaborators have been establishing a new approach called the *distributed time-triggered simulation* (DTS) *scheme* which is conceptually simple and easy to use but widely applicable. The concept was initiated in the course of developing a new-generation object-oriented real-time programming scheme called the *time-triggered message-triggered object* (TMO) programming scheme. Some fundamental issues inherent in distributed real-time simulation and major design principles and implementation techniques for resolving those issues within the DTS framework are presented. This technical foundation has been identified through multiple major DTS experiments conducted over the years. Some issues that require further research to realize the full potentials of the DTS scheme are also discussed.

Keywords: real time, embedded, clock, tick, simulation, update, dependency, DTS, TMO, time triggered, message, object, middleware, distributed, parallel, programming.

1. Introduction

Signs are everywhere to indicate that *object-oriented (OO) real-time (RT) distributed computing* has become a rapidly growing branch of computer science and engineering [IEE00, ISO98, ISO99, ISO00, OMG00]. Its growth is fueled by the strong needs present in industry for the RT programming methods and tools which will bring about orders-of-magnitude improvement over the traditional RT programming practiced with low-level programming languages (C or assembly programming languages) and styles.

Among several branches of this young OO RT distributed computing field, a less developed branch which is in its infancy but possesses huge potential markets is *OO distributed RT simulation*. Here *RT simulation* refers to the accurate mode of simulation in which the simulator components (or *simulator objects*) show the timing behavior that are the same as or similar to the timing behavior of the *simulation targets*.

RT simulator developments are under increasing demands [Kim96]. For example, continuing advances in virtual reality applications accompany increasing demands for more powerful RT simulation capabilities. Numerous other examples can also be found in the RT computer control field. Not only description but also simulation of application environments is often performed as integral steps of validating control computer system designs [Ell93, Guy93, Kim97, Zei93]. Here RT simulators of application environments can often enable highly cost-effective testing of the control computer systems implemented. Such testing can be a lot cheaper than the testing performed in actual application environments while being much more effective than the testing based on non-RT simulators of environments.

As the complexities of RT simulators grow, the use of distributed / parallel RT simulation approaches become imperative. However, practical distributed RT simulation techniques have not been established in sufficiently reliable forms for industrial use.

In fact, even in cases of RT simulation which uses a single computing node, there is a lot to be desired in the industrial state of the art. A approach to model the simulation targets, especially those which enable *multi-fidelity representation of the timing behavior of the simulation targets*, is highly desired.

Since early 1990's this author and his collaborators have been establishing a new-generation OO RT programming scheme called the *time-triggered message-triggered object* (TMO) programming scheme [Kim94, Kim97, Kim99a, Kim00b]. In the course of developing an RT system engineering methodology based on this TMO programming scheme, a new approach to RT simulation which is conceptually simple and easy to use but widely applicable, has also been established [Kim94, Kim96b, Kim99c]. This approach called the *distributed time-triggered simulation* (DTS) *scheme* is an attractively simple approach to parallel and distributed RT simulation.

The TMO is a natural, syntactically minor, and semantically powerful extension of the conventional object(s) [Kim97, Kim99a, Kim00b]. The TMO is capable of representing uniformly and with variable degrees of precision both RT embedded computer systems and their application environments.

0-7695-1089-2/01 $10.00 © 2001 IEEE

The key idea of DTS is to let distributed simulator nodes, or more generally, *simulator objects*, advance to the next simulation step simultaneously when the globally available RT clock reaches a certain time-point. Numerous message exchanges required in previously studied distributed simulation approaches in order to keep the simulator nodes properly synchronized in advancing to the next simulation step are obviated in DTS.

While the basic framework of the DTS scheme is simple, attempts to incorporate optimal implementation techniques into DTS have revealed some fundamental challenging issues inherent in distributed RT simulation. Some major design principles and implementation techniques which resolve such issues in constructing DTS object networks have been identified through multiple DTS experiments conducted over the years. Therefore, the main goal of this paper is to summarize such fundamental issues and some techniques for resolving the issues within the DTS framework. Some issues that require further research to realize the full potentials of the DTS scheme are also discussed.

In the next section, major challenges in developing distributed RT simulation technologies are discussed along with the core underlying concept of the DTS scheme. Section 3 provides an overview of the TMO programming scheme. Then in Section 4, the DTS scheme facilitated by the TMO programming scheme is reviewed first and the issues in realizing effective distributed RT simulators as well as some resolution techniques which can be incorporated into the DTS scheme are discussed. The paper concludes in Section 5.

2. Main challenge in distributed RT simulation and the DTS framework

In a computer-based simulation, an integer called the *simulator clock* is used and each new incremented value of the simulator clock drives new simulation activities (a new *simulation step*). Since an RT simulator must exhibit the timing behavior which is the same as or very close to that of the simulation target, the *simulator clock* must "tick" at a *steady rate*. Each tick of the simulator clock is commenced and administered by referencing an RT clock in the *simulation execution engine* (a computer system running the simulation program). The *ticking rate* of the simulator clock in an RT simulator must be chosen with the following understanding:

> For all (real-time) events which should be simulated during any *ticking interval* of the simulator clock, the exact microscopic timings of those events may be transparent to the user of the RT simulator.

Only the resulting state of the simulator at the end of the ticking interval may be seen by the user. That is, only the fact of whether each such event occurred or not at some (unknown) point during the interval should be of interest to the user. This requirement is called *the simulator clock atomicity requirement* [Kim99c]. All computational activities taking place during a ticking interval of the simulator clock may be viewed as one *simulation-step*.

As an illustration of the relationship between the ticking rate and the simulation precision, consider the case of simulating cars moving on a freeway. Since the moving pattern of each car is affected by those of the cars moving 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 car will be updated every 5 seconds. However, if a typical car can change its lane in one second and a lane change by a car can have big impacts on subsequent behaviors of some other cars, updating the states of all cars every 5 seconds means a very low-fidelity *low-precision* simulation.

To be more specific, assume that car collisions involving a car changing a lane are to be dealt with. It is then possible that the simulator user may see the result of a collision occurring at the time corresponding to 0.1 seconds after a certain tick in one simulation run whereas in another run of the same simulator, the user may see the result of a collision occurring at the time corresponding to 4.9 seconds after the same tick, which may be a drastically different result, depending on the play of the random number generator. In the former case, the collision can impact the behavior of the cars in collision and the nearby cars during the remaining 4.9 seconds. This makes it necessary to re-simulate the activities of certain cars if those cars were simulated before the "discovery" of the collision and they were in positions to be affected by the collision during the current 5-seconds interval.

These multiple re-simulations of cars during one simulation step can degrade the performance of the simulator down to an unusable level. Therefore, in general, multiple re-simulations of the same simulation target in one simulation step must be avoided. This then leads to the simulator designer to adopt a simulation model which uses the same state-update logic independent of the ticking rate of the simulator clock. This is why the precision of a simulator becomes low in practice when the ticking rate of the simulator clock is low or equivalently, the ticking interval becomes large.

On the other hand, if the states of all (say, 10000) cars were to be updated every 50 milliseconds for the sake of realizing high-precision RT simulation, then the *simulation model* of the cars and the freeway may lead to an excessive computational load imposed on the *execution engine*. Therefore, the ticking interval of the simulator clock cannot be made indefinitely small.

In distributed RT simulation, simulator objects are distributed among multiple nodes. This is a compelling mode of simulation when a large-scale simulation model is used. *Synchronization of the simulation-steps of distributed simulator objects* is then a key challenge. In

other words, a simulation-step executed by every member of the distributed simulator object group must be synchronized with the corresponding simulation-step executed by any other member. This also means that the simulator clock for one simulator object must commence the n-th tick neither before the (n-1) - th tick by the clock for another simulator object nor after the (n+1) - th tick by the clock for another simulator object.

Therefore, every member must perform some activities necessary to stay synchronized with other members. For example, distributed nodes may exchange completion reports at the end of each simulation-step. However, this is not an efficient approach when the number of nodes used is large. The essence of the *distributed time-triggered simulation* (DTS) approach is the following:

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

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

The DTS approach has major advantages over other distributed simulation approaches, even if we assume that the latter approaches can be adapted somehow to enable RT simulation [Fuj90, Mis86]. This is because synchronization of simulation-steps executed by distributed simulator objects under the DTS scheme does not require message exchanges among the host nodes (not counting the message exchanges which may be needed at a certain low frequency for resynchronizing the RT clocks of the nodes). The advantages become decisive in heavy-load distributed simulation situations.

However, even with the DTS approach, exchanges of messages that represent movements of certain simulation targets from the territory covered by one simulator component (e.g., TMO) to the territory covered by another simulator component are inevitable. Therefore, the ticking interval must be long enough to cover this kind of message exchanges.

3. An overview of the TMO scheme

The *time-triggered message-triggered object* (TMO) scheme was established in early 1990's [Kim94, Kim97, Kim99a, Kim00b] with a concrete syntactic structure and execution semantics for economical reliable design and implementation of RT systems. The TMO programming scheme is a general-style component programming scheme and supports design of all types of components including distributable hard-RT objects and distributable non-RT objects within one general structure.

TMOs are devised to contain only high-level intuitive and yet precise expressions of timing requirements. No specification of timing requirements in (indirect) terms other than *start-windows* and *completion deadlines* for program units (e.g., object methods) and *time-windows*

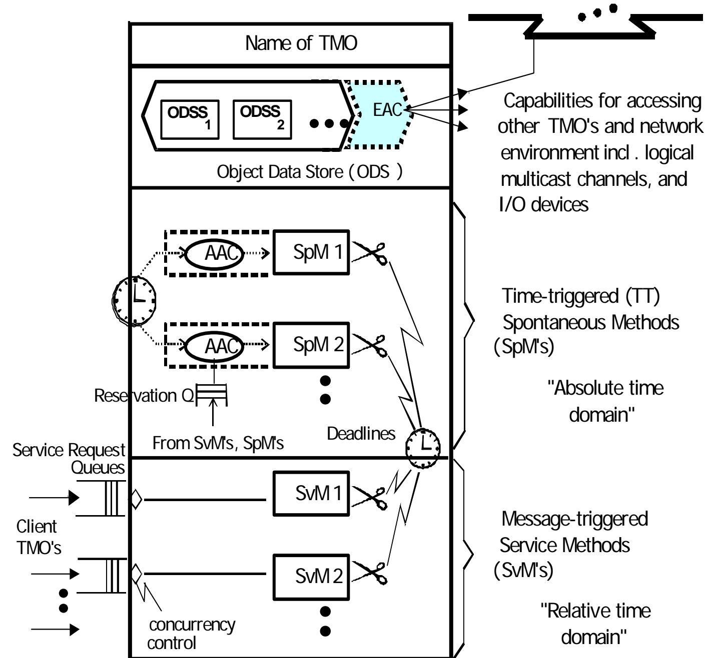


Figure 1. The basic structure of TMO (Adapted from [Kim97])

for output actions is required. For example, priorities are attributes often attached by the OS to low-level program abstractions such as threads and they are not natural expressions of timing requirements. Therefore, no such indirect and inaccurate styles of expressing timing requirements are associated with objects and methods [Kim94, Kim99a, Kim00b].

At the same time the TMO scheme is aimed for enabling a great reduction of the designer's efforts in guaranteeing timely service capabilities of distributed computing application systems. It has been formulated from the beginning with the objective of enabling *design-time guaranteeing of timely actions*. The TMO incorporates several rules for execution of its components that make the analysis of the worst-case time behavior of TMOs to be systematic and relatively easy while not reducing the programming power in any way [Kim97, Kim99a].

3.1 TMO structure and design paradigms

TMO is a natural, syntactically minor, and semantically powerful extension of the conventional object(s) [Kim97, Kim99a, Kim00b]. As depicted in Figure 1, the basic TMO structure consists of four parts:

ODS-sec = object-data-store section: list of *object-data-store segments* (ODSS's);

EAC-sec = *environment access-capability* section: list of *gates* to remote object methods, logical communication channels, and I/O device interfaces;

SpM-sec = *spontaneous-method* section: list of *spontaneous methods*;

SvM-sec = *service-method* section.

Major features are summarized below. The second and third are the most conspicuous unique extensions of conventional object(s).

(a) Distributed computing component:

The TMO is a distributed computing component and thus TMOs distributed over multiple nodes may interact 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 *SpMs*), which are clearly separated from the conventional *service methods* (*SvMs*). The SpM executions are triggered upon reaching of the real-time 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 SpMs.

(c) Basic concurrency constraint (BCC):

This rule prevents potential conflicts between SpMs and SvMs and reduces the designer's efforts in guaranteeing timely service capabilities of TMOs. 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 an execution time-window big enough for the SvM that does not overlap with the execution time-window of any SpM which accesses the same ODSSs to be accessed by the SvM, opens up. However, the BCC does not stand in the way of either concurrent SpM executions or concurrent SvM executions.

(d) Guaranteed completion time and deadline:

The TMO incorporates deadlines 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.

Triggering times for SpMs must be fully specified as constants during the design time. Those RT 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" }

An underlying design philosophy of the TMO scheme is that an RT computer system will always take the form of a network of TMOs. The designer of each TMO provides a guarantee of timely service capabilities of the object. The designer 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* (GCT) 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* (a composition of hardware, node OS, and middleware) available, the server object can be implemented 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.

3.2 Middleware supporting OO RT program

A cost-effective way to support execution of OO RT distributed programs is to realize an execution engine by developing middleware running on well established commercial software / hardware platforms. Middleware which together with node OSs and hardware make up TMO execution engines, have been developed [Kim96a, Kim99b, Kim99d].

To be more specific, an efficient middleware architecture, named *TMO Support Middleware* (TMOSM), has been developed. Then a prototype implementation on Windows NT, TMOSM/NT, has been obtained [Kim99b] (available from http://dream.eng.uci.edu). Our experiences indicate that even this middleware extension of a general-purpose OS (Windows NT) can support application actions with the 10ms-level timing precision. Also, another prototype implementation in the form of a CORBA service, TMOSM/AnyORB/NT, that runs on platforms equipped with Windows NT and a basic object request broker (ORB) and supports CORBA-compliant application TMOs has been obtained [Kim99d].

TMOSM implementations on the Windows NT platforms have been operational in the author's laboratory and other laboratories along with several non-trivial applications structured as TMO Networks. A friendly application programming interface (API) wrapping the services of TMOSM has also been developed and named the *TMO Support Library* (*TMOSL*) [Kim99b, Kim00b]. It consists of a number of C++ classes. TMOSL empowers C++ programmers with powerful and natural mechanisms for specification of unique and essential features of RT distributed programs.

3.3 TMO structuring in environment modeling and multi-step multi-level design and implementation

The attractive basic design style facilitated by the TMO structuring is to produce a network of TMO's

meeting the application requirements in a top-down multi-step fashion [Kim97]. The engineering of an application system can start with a single TMO representation of the entire application environment (including the computer system to be designed) and proceeds through step-by-step expansion of the initial single TMO model toward a final implementation in the form of a network of TMO's executing on engines. This top-down process can also produce an RT simulator of the application environment, again in the form of a TMO network performing DTS.

4. TMO-structured distributed time-triggered simulation (DTS)

The DTS approach facilitated by the TMO programming scheme uses distributed TMOs of which SpMs execute simulation-steps [Kim94, Kim96b, Kim97, Kim99c]. This systematic approach has been shown to be practical via several experiments which dealt with RT distributed applications such as a missile defense command-control application [Kim97, Sho98], a freeway car traffic control application [Kim99c], and a steel mill factory control application [Kim00a].

For example, a freeway-segment can be represented at a high level and simulated by the TMO in Figure 2.

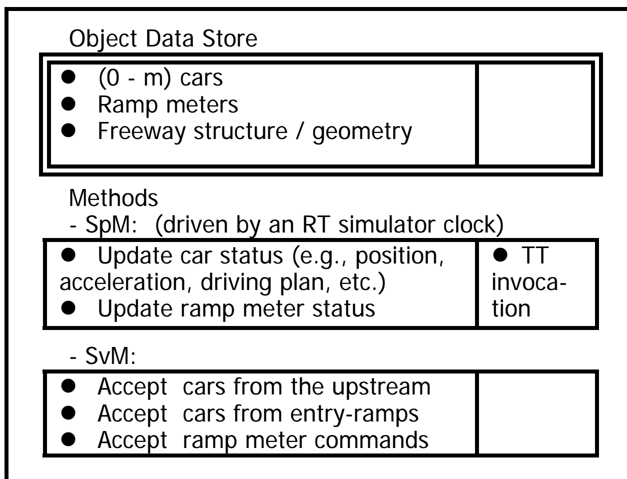


Figure 2. A TMO-structured simulation model for a freeway-segment

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 or SpM, when executed, updates a variable-set in the ODS representing the state of some simulation target item (i.e., physical item such as car, ramp meter, etc) to reflect the current state of the target item. Ideally the TT methods should be *activated continuously* and each of their executions be *completed instantly*. However, the limited power of the execution

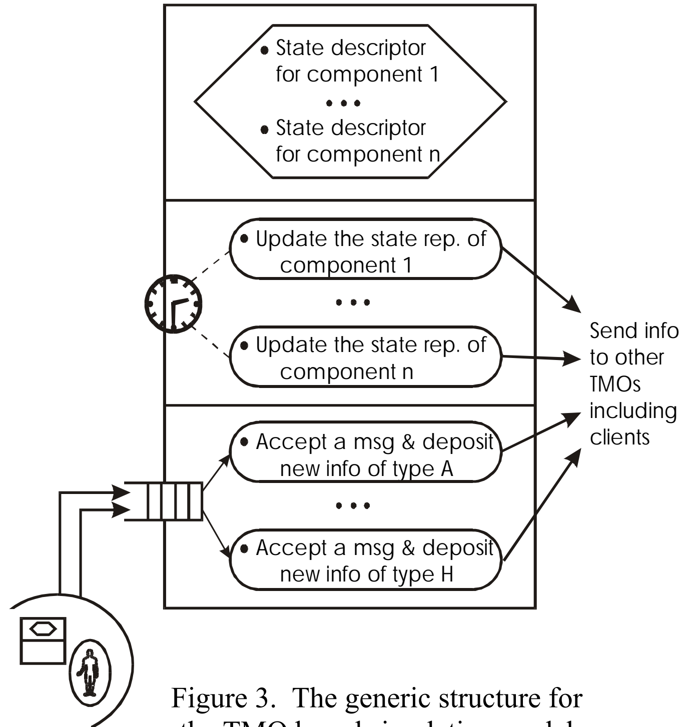


Figure 3. The generic structure for the TMO based simulation model

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 target item simulator clock*. Each execution of a TT method must be completed within one ticking interval of the target item simulator clock. Therefore, TT methods are the mechanisms for approximately simulating continuous state changes that occur naturally in the target items in the environment.

The natural parallelism that exists among the simulation target items in the environment can be precisely represented by use of multiple TT methods which may be activated simultaneously. An alternative, which sometimes leads to a more efficient but maybe less easily understandable implementation, is to have a single TT method update the state variables representing multiple heterogeneous types of simulation target items. This alternative is the normal choice when the TMO execution engine contains only one CPU. In general, the precision 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 target item simulator clocks).

Figure 3 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 and broad applicability of the structure depicted in Figures 2 and 3, are just some of many attractive features of the TMO-structured DTS scheme that offers significant potential for improving the economy and efficiency of distributed / parallel RT simulation over the state of the art. Other major features include: 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.

5. Challenging issues in efficient implementation of distributed time-triggered simulation (DTS)

In this section, some fundamental challenging issues in realizing efficient distributed RT simulators are discussed. Some techniques for resolving the issues within the DTS framework are also presented.

5.1 Group server and monitor (GSM) object

Suppose that (0 - m) cars in the ODS in Figure 2 are replaced by (0 - m) airplanes, Ramp meters by (0 - n) ships, and the Freeway structure / geometry by air-sea theater space. Also assume that the TT methods in Figure 2 are replaced by appropriate TT methods. The resulting single TMO representation, which may be called the Theater TMO, can be expanded into a more detailed representation structured in the form of a network of TMOs, each representing an airplane or ship, as shown in Figure 4.

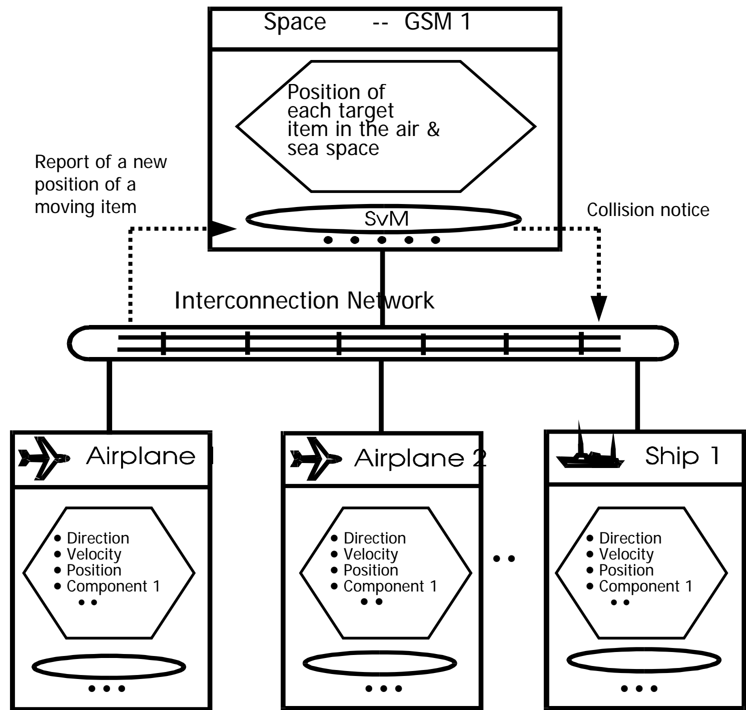


Figure 4. An expanded network of TMOs representing the simulation target in more details (Adapted from [Kim96b])

Figure 4 also depicts an interesting role played by the TMO representing the air and sea space. This Space TMO maintains information on how the space is occupied. It facilitates detecting collisions between moving items such as airplanes, etc. As the TMO structured simulator of an airplane progresses after each simulator clock tick, the position of the airplane is updated and recorded within the TMO. In addition, this TMO notifies the Space TMO of the new position of the former. The latter TMO in turn checks if the airplane has now collided with any other environment item such as an airplane, ship, etc., and, if so, notifies the TMOs simulating the collided items. The notified TMOs then simulate the post-collision behavior of their simulation targets starting with the following tick of the simulator clock.

In a sense, the Space TMO supports the TMOs simulating the moving environment items. Therefore, it is called a *group server and monitor* (GSM) TMO. Each ticking interval of the simulator clock must cover the time spent in interaction between a GSM TMO(s) and monitored TMOs. Often the GSM TMO may contain service methods (SvMs) only, i.e., no TT methods. If the number of moving items is large, then multiple GSM TMOs, each supporting a different group of TMOs simulating the physical environment items, can be utilized.

On the other hand, the fact that there is the possibility of a collision among airplanes, means that the state descriptors for airplanes are tightly coupled in nature. In most cases it is not worth decomposing a TMO containing such tightly coupled data members into a group of TMOs supported by a GSM TMO. In the case of the simulation in Figure 4, such decomposition can be justified only if each update of the state descriptor for an airplane and that for a ship is highly time-consuming and thus parallel updating of multiple Airplane TMOs and Ship TMOs is really necessary. Without decomposition, parallel updating can be done if the computing node executing the monolithic Theater TMO has a sufficient number of processors and multiple TT methods designed to update state descriptors of different airplanes and ships. With decomposition, parallel updating could be done by either using a multi-processor based TMO execution engine or distributing Airplane TMOs and Ship TMOs to different nodes. However, the speed gain from such parallel updating should be measured against the overhead incurred in interactions between the GSM TMO and Airplane TMOs and Ship TMOs.

It may also be possible to exploit parallelism inside each Airplane TMO and each Ship TMO. If the speed gain achievable through such parallelism exploitation is

substantial, then it becomes an added incentive for decomposing the Theater TMO which involves creation of GSM TMO(s).

5.2 Update-dependency among simulator objects

Figure 5 shows the simulated states of cars at 10:00:00 am and at 10:00:01 am, respectively. Let X(10:00:00) denote the simulated state of car X at 10:00:00 am. The terms "simulated car", "car simulator", "car simulator object", and "car" are used interchangeably whenever there is no ambiguity. Assume that the entire simulator is a single TMO and thus runs on one node.

Therefore, at 10:00:01 am, the simulator must update the states of the cars as shown in Figure 5, using the information on the states of the cars at 10:00:00 am. Each car is characterized by the parameters such as (i) the *speed change probability* represented in the form of a function of the distance from the car to each adjacent car, the current speed, the preferred speed, and the road condition, (ii) *the lane change probability* represented as a function of the distance to the target exit and the distance to each adjacent car, etc. Random numbers are used in each update.

In updating car D to D(10:00:01), not only D(10:00:00) but also at least B(10:00:00), C(10:00:00), E(10:00:00), and F(10:00:00) must be used. In principle, cars can be updated in any order since in updating each car, only the old states (i.e., the states at 10:00:00) of adjacent cars and other parts of the freeway are used.

Actually it is not that simple. D(10:00:01) includes a new position of car D at 10:00:01. The new position can fall within a certain range depending upon the value of the random number used and other parameters such as possible speed range, etc. Suppose all cars have been independently updated to the states effective at 10:00:01. It is then possible for a car, say D, to be positioned ahead of its previous predecessor, say B, at 10:00:01. This can occur due to the effects of random numbers. However, this new state of the simulator is an *inconsistent state* (since D could not have flown over B).

This means that either D(10:00:01) or B(10:00:01) must be corrected. For one of the two cars, simply the current state value for 10:00:01 is discarded and a new state value for 10:00:01 is calculated. The new state value is then checked whether it leads the entire simulator to a consistent sate.

Therefore, it is better to sort cars and update them in the sorted order. If the order is to update front cars first and rear cars later, then whenever a value for the new state of a car is calculated, a check is made whether it is *in conflict* with the already calculated new states of the cars in the front. If a conflict is detected (i.e., if the value is such that it leads the simulator to inconsistency), the value is discarded and a try is made to produce a new value until a conflict-free value is produced. When two simulator objects, e.g., car simulators B and D, are in

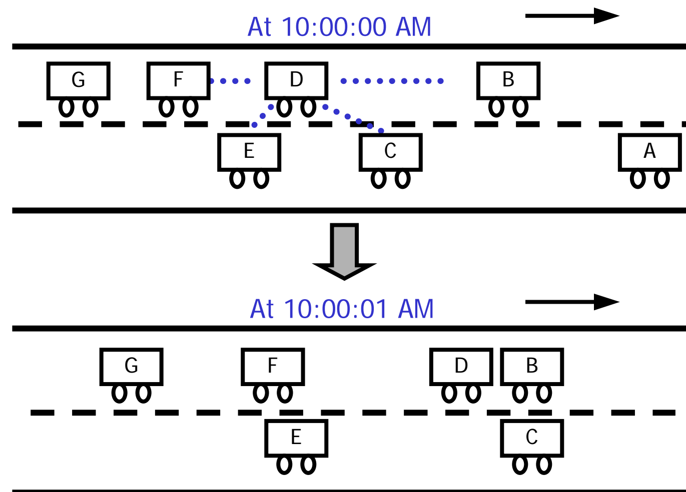


Figure 5. Single-node simulation of cars

such relationship and thus cannot be updated independently, they are said to be *update-dependent* upon each other.

On the other hand, suppose the distance between car D and any of its adjacent cars at 10:00:00 is greater than the maximum distance that car D can travel in one ticking interval, i.e., one second. Then, there is no possibility for a newly calculated state value for D being in conflict with a newly calculated state value for any of D's adjacent cars. This means that D can be updated independently of its adjacent cars. Therefore, the *update-dependency* in the freeway simulation situation is a function of (i) the ticking rate (or the length of the ticking interval), (ii) the speeds of cars, and (iii) the distances between cars, among others. To put it another way, the update-dependency can be "broken" by changing any of those parameters. Note that *the update-dependency is a transitive relation*. Of those parameters, the ticking rate is the only basic parameter common to all simulators.

5.3 Minimization of the impacts of update-dependency among distributed simulator objects

Suppose now a distributed simulation is attempted by partitioning the simulator (TMO) covering the freeway in Figure 5 into multiple simulator subsystems (TMOs),

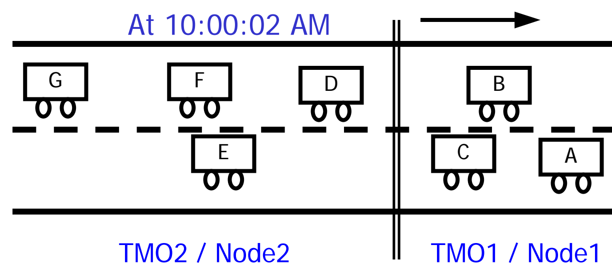


Figure 6. Distributed simulation of cars

each covering a segment of the freeway, and running each simulator subsystem on a separate node. Figure 6 illustrates a two-TMO DTS system. As soon as the front edge of a car covered by TMO2 reaches the boundary between the two freeway-segments, the car should be taken over by TMO1.

An important aspect to note here is that the update-dependency among cars does not change by this partitioning of the simulator into multiple simulator subsystems. Suppose that in Figure 6, both TMOs run in parallel and each TMO updates front cars first and rear cars later. Cars are to be updated to the states effective at 10:00:03. If car D is update-dependent on car B and car C, then TMO2 needs new state values, B(10:00:03) and C(10:00:03), calculated by TMO1 in order to establish D(10:00:03).

If TMO2 must wait until TMO1 completes updating cars B and C before it can update car D, then almost no parallelism remains between TMO1 and TMO2. Therefore, such distributed simulation is *worse* than single-node simulation ! An innovative approach is needed here to resolve this challenging situation.

When TMO1 updates car B from B(10:00:02) to B(10:00:03), it uses primarily
(1) a random number α(B, 10:00:03) in addition to
(2) B(10:00:02) which is the state of B valid at the last clock tick,
(3) the states of adjacent cars valid at the last clock-tick (i.e., A(10:00:02) and C(10:00:02)), and
(4) the new states of B's update-dependent neighbor cars that have just been computed in the current simulation-step (i.e., A(10:00:03)).

One can then note the following. TMO2 can produce B(10:00:03) *in the shadow* while TMO1 produces the same B(10:00:03) under the following conditions:

(C1) TMO1 sends to TMO2 all of its (TMO1's) state values updated during the last simulation-step including A(10:00:02), B(10:00:02), and C(10:00:02) before the arrival of the clock tick 10:00:03; and

(C2) TMO1 generates the random numbers to be used in the simulation-step starting at 10:00:03, i.e., α(A, 10:00:03), α(B, 10:00:03), and α(C, 10:00:03), in advance and send them to TMO2 before the arrival of the clock tick 10:00:03.

(C3) If TMO1 uses information from an outside source, e.g., another TMO simulating surface streets, the same message carrying such information arrives at both TMO1 and TMO2 before the time at which TMO1 uses the information.

If TMO2 can produce B(10:00:03) and C(10:00:03) in the shadow without waiting for TMO1 to generate them, then TMO2 can proceed to update car D and other cars in its territory. Therefore, a certain amount of parallelism can still be exploited between TMO1 and TMO2 without sacrificing the *simulation consistency* i.e.,. the property of a simulation being performed in a consistent manner. This approach is called the *primary-shadow state update approach*.

The second condition C2 and the third condition C3 are easy to meet. The first condition C1 presents a challenge for its optimal implementation. First, TMO2 needs to get updated state values only for those cars in TMO1 upon which front-end cars in TMO2, D and E, are update-dependent. This cuts down the costs of duplicated state updating and message communication from TMO1 to TMO2. Again, one should remember that the update-dependency is a transitive relation and that some cars in TMO2 can become update-dependent on some in TMO1 which the former cars were not previously update-dependent on. On the other hand, if front-end cars in TMO2 are update-dependent on all the cars in TMO1, no reduction of duplicated state updating is possible and thus such situations are not suitable for distributed simulation. Therefore, it is worth for both TMO1 and TMO2 to *keep track of update-dependency developments and disappearances*.

Secondly, all the cars currently in TMO1 were in TMO2 for a while. Therefore, as long as TMO1 generates all the random numbers and sends them to TMO2 in advance, then TMO2 can continue to update the copies of the cars in the shadow after it sends the cars to TMO1. Such copies kept in TMO2 of the cars in TMO1 are called the *shadows* of the latter cars. Exploitation of this property results in elimination of the messages from TMO1 to TMO2 that carry state values for the cars in TMO1. However, under this approach, it is difficult to make TMO2 to keep the shadows of only a subset of the cars in TMO1.

Therefore, the key factor that determines the efficiency of distributed / parallel RT simulation is the update-dependency. Again, the update-dependency can be broken by increasing the ticking rate of the simulator clock but there is a limit on the ticking rate which can be supported by a given simulator execution engine.

5.4 Safe reduction of the redundant state updating in the shadow at the compromise of simulation consistency

If the amount of redundant state updating in the shadow needs to be reduced, one can do it at the compromise of *simulation consistency* but without introducing *distributed state inconsistency*. In the case of Figure 6, the approach is to reduce some front-end shadows in TMO2. For example, assume that car D is update-dependent on cars A, B, and C. Then normally shadows of A, B, and C are maintained in TMO2. An option for TMO2 here is to discard the shadows and assume pessimistically that those cars (A, B, and C) will move for "minimal distance". This assumption is far from the truth since TMO1 will most probably move those cars for more than minimal distance. Nevertheless, TMO2 can update the cars in its territory based on the pessimistic

assumption. Here no conflicts can develop between TMO1 and TMO2. Just some degree of simulation consistency has been lost in that in a single-node simulator, car D and other following cars would have moved for larger distance.

After removing the shadows, if TMO2 needs to create shadows again, then it needs to get the full state values on those cars from TMO1. However, the lost simulation consistency cannot be compensated. Moreover, repeated loss of the simulation consistency leads inevitably to the loss of synchronization accuracy.

6. Remaining issues

6.1 Ticking rate and the simulator execution engine

The precision of the RT simulator is proportional to the ticking rate. In addition, for efficient distributed simulation, minimizing the update-dependency is necessary. This update-dependency can also be broken by increasing the ticking rate. Yet the ticking interval must be long enough to accommodate necessary message communications. Therefore, it is desirable to use computing platforms yielding small message delays. Use of highly parallel computing platforms is an attractive approach. DTS using such platforms is considered a timely and potentially fruitful area for future research.

If distributed computing nodes used in DTS are connected via bus LANs such as Ethernet, then it pays off to do some careful scheduling of bus access by the nodes. This is because without careful scheduling, the probability of a collision occurrence in bus access is very high, especially during the phase for exchanging state descriptors for some moving items in each simulation-step. Such scheduling is the job for the simulation execution engine consisting of distributed computing nodes. .

6.2 Use of logical multicast channels

Another issue to be investigated in future research is the possible use of logical multicast channels instead of remote method calls for passing some state descriptors for simulation target items between TMOs. If simulator TMOs are made to interact via logical multicast channels such as real-time memory replication and multicast channels (RMMCs) [Kim00b], then the TMOs become somewhat more loosely coupled. This could lead to less overhead but this potential needs to be confirmed via both analytical and experimental research.

7. Conclusion

The DTS scheme is a fundamentally new type of an approach to distributed RT simulation. It requires a global time base which provides consistent RT values to application software running on distributed nodes. The developments of clock synchronization mechanisms based on GPS (global positioning system) or LAN-based hardware support in the past decade have made the high-precision global time base easily affordable to most system developers [Kop97]. Although it has many advantages over other proposed distributed RT simulation approaches, the exploitation of its full potential requires advanced computing platforms such as highly parallel computing platforms.

Some practical DTS TMOs have been implemented in the author's and other laboratories [Kim97, Kim99c, Kim00a, Sho98]. However, not all the optimization techniques discussed in Section 5 have been incorporated and some of those are currently being incorporated.

The DTS scheme is a young research subject and thus much more research, especially, experimental research with advanced distributed and parallel computing platforms, is needed to learn better about its full potential and costs.

Acknowledgements: 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 NSF Next-Generation Software (NGS) Program under Grant 99-75053, and in part by the NSF Information Technology Research (ITR) program under the grant No. 00-86147.

References

[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.

[IEE00] '*A special issue of Computer (a magazine of IEEE Computer Society) on Object-oriented Real-time distributed Computing*', June 2000.

[ISO98] *Proc. ISORC '98 (IEEE CS 1st Int'l Symp. on Object-oriented Real-time distributed Computing, Kyoto)*, IEEE CS Press, April 1998.

[ISO99] *Proc. ISORC '99 (IEEE CS 2nd Int'l Symp. on Object-oriented Real-time distributed Computing, St. Malo)*, IEEE CS Press, May 1999.

[ISO00] *Proc. ISORC 2000 (IEEE CS 3rd Int'l Symp. on Object-oriented Real-time distributed Computing, Newport Beach)*, IEEE CS Press, March 2000.

[Kim94] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. WORDS '94 (IEEE CS '94 Work. on Object-Oriented Real-Time Dependable Systems)*, Oct. 1994, Dana Point, pp.36-45.

[Kim96a] Kim, K.H. et al., "The DREAM Library Support for PCD and RTO.k programming in C++", *Proc. WORDS '96*, Laguna Beach, Feb. 96, pp. 59-68.

[Kim96b] Kim, K.H., Nguyen, C., and Park, C., "Real-Time Simulation Techniques Based on the RTO.k Object Modeling", *Proc. COMPSAC '96 (IEEE CS '96 Software & Applications Conf.),* Seoul, August 1996, pp. 176-183.

[Kim97] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No.8, August 1997, pp. 62-70.

[Kim99a] Kim, K.H., "Real-Time Object-Oriented Distributed Software Engineering and the TMO Scheme", *Int'l Jour. of Software Engineering & Knowledge Engineering*, Vol. 9, No.2, April 1999, pp.251-276.

[Kim99b] Kim, K.H. Ishida, Masaki, Liu, Juqiang, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. ISORC '99* (*2nd IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing*), St. Malo, France, May, 1999, pp.54-63.

[Kim99c] Kim, K.H., Liu, J., and Ishida, M., "Distributed Object-Oriented Real-Time Simulation of Ground Transportation Networks with the TMO Structuring Scheme", *Proc. COMPSAC '99* (IEEE CS Computer Software & Applications Conf.), Phoenix, AZ, Oct. 1999, pp.130-138.

[Kim99d] Kim, K.H., Liu, J.Q., Miyazaki, H., and Shokri, E.H., "A CORBA Service Enabling Programmer-Friendly Object-Oriented Real-Time Distributed Computing", *Proc. of WORDS '99F (IEEE CS 5th Workshop on Object-oriented Real-time Dependable Systems)*, Monterey, Nov. 1999, pp.101-107.

[Kim00a] Kim, K.H., "Real-time Object-oriented Distributed Computing", in John G. Webster ed., '*Encyclopedia of Electrical & Electronics Engineering*', Supplementary Volume, John Wiley & Sons, 2000.

[Kim00b] Kim, K.H., "APIs Enabling High-Level Real-Time Distributed Object Programming", *IEEE Computer*, June 2000, pp.72-80.

[Kop97] Kopetz, H., '*Real-Time Systems: Design Principles for Distributed Embedded Applications*', Kluwer Academic Pub., ISBN: 0-7923-9894-7, Boston, 1997

[Mis86] Misra, J., "Distributed Discrete-Event Simulation", ACM Computing Surveys, Vol. 18, No. 1, March 1986, pp.39-65.

[OMG00] 'Collection of Slide Presentations, 1st OMG Workshop on Real-Time / Embedded Distributed Object Computing', July 2000, Crystal City, VA.

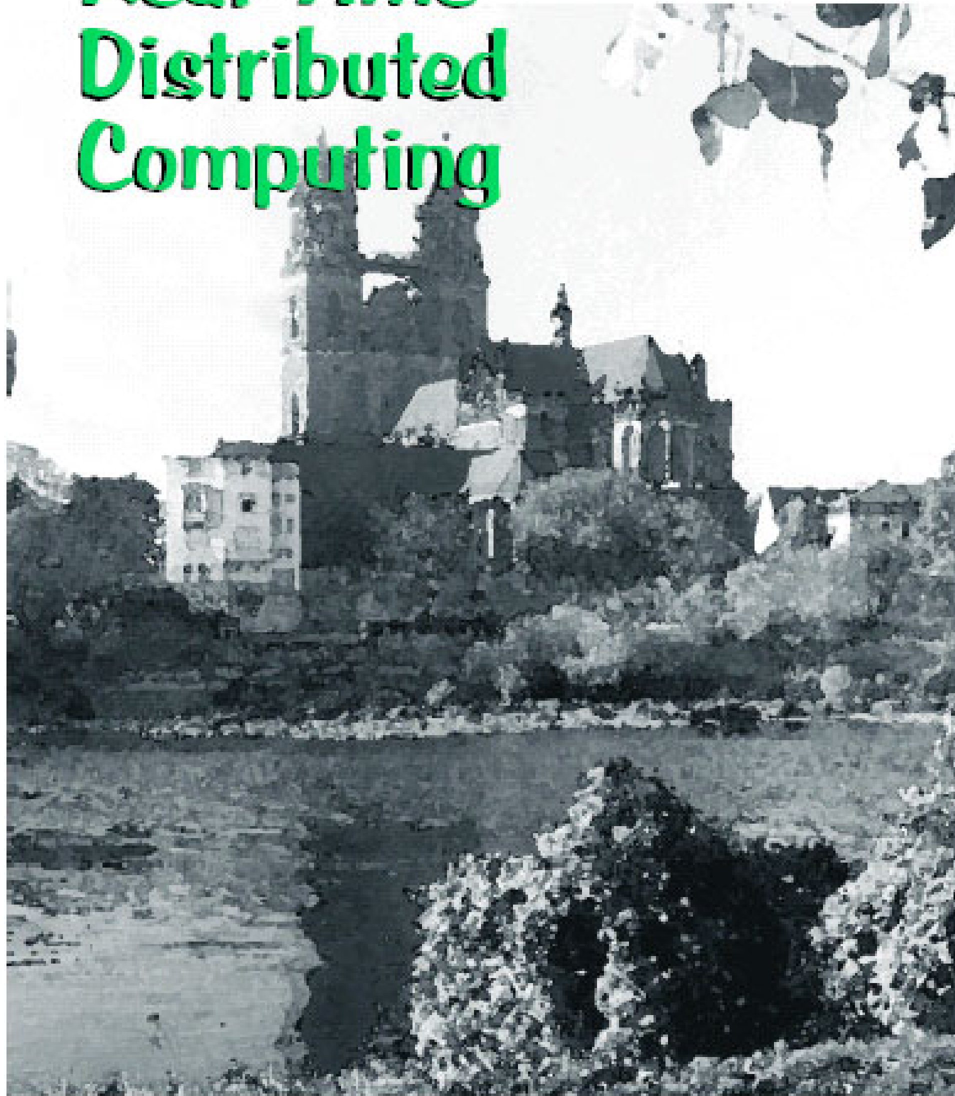
[Sho98] Shokri, E., Crane, P., and Kim, K.H., "An Implementation Model for Time-Triggered Message-Triggered Object Support Mechanisms in CORBA-Compliant COTS Platforms", *Proc. ISORC '98 (IEEE CS 1st Int'l Symp. on Object-oriented Real-time distributed Computing)*, Kyoto, Japan, April 1998, pp. 12-21.

[Zei93] Zeigler, B., and Kim, J., "Extending the DEVS-Scheme Knowledge-Based Simulation Environment for Real-Time Event-Based Control", *IEEE Trans. on Robotics and Automation*, Vol.9, No.3, 6/93, pp.351-356.

Proceedings

The Fourth IEEE International Symposium on

Object-Oriented Real-Time Distributed Computing



Magdeburg, Germany
2-4 May 2001

ISORC 2001

Sponsored by
IEEE Computer Society Technical Committee on Distributed Processing
In cooperation with
IFIP WG. 10.4
OMG
Otto-von-Guericke-Universität Magdeburg