

Copyright 1996 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

## Real-Time Simulation Techniques Based on the RTO.k Object Modeling

K. H. (Kane) Kim

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

Cuong Nguyen  
USN Naval Surface Warfare Center, U.S.A.

and

Chanmo Park  
Postech, Korea

**Abstract:** Real-time simulation is an advanced mode of simulation in which the simulation objects are designed to show the same timing behavior that the simulation targets do. A new approach to real-time simulation which is based on the RTO.k object modeling is discussed in this paper. The RTO.k object, which is a real-time extension of the well-established object structure, is capable of uniformly and accurately representing both real-time embedded computer systems and application environments. This simulation approach has many attractive features, e.g., expandability, modifiability, adaptability for efficient parallel processing, etc. In spite of its promising nature, the approach is an immature one in many respects and some desirable directions for future work aimed toward maturing the technology are also discussed.

**Index Terms:** real-time simulation, real-time object, simulator clock, parallel processing, distributed computing, real-time clock, RTO.k

### 1. Introduction

An accurate mode of simulation called the real-time simulation in which *the simulation objects show the same timing behavior that the simulation targets do*, is under increasing demands. For example, continuing advances in virtual reality applications accompany increasing demands for more powerful real-time simulation capabilities. Numerous other examples can also be found in the real-time computer control field. Not only description but also simulation of application environments is often performed as integral steps of validating control computer system designs [El193, Guy93, Kim94b, Zei93]. A desirable mode of simulating application environments in such situations is the real-time simulation.

In order to support such real-time simulation, new approaches to model the simulation targets, especially those which enable multi-fidelity representation of the timing behavior of the simulation target, are needed. In our view, a preferred modeling approach for use in real-time simulation should be of object-oriented (OO) type, considering the modularity, generality, and natural abstraction benefits that OO approaches bring in.

Since existing object models do not possess adequate capabilities for representing the timing behavior accurately, we are forced to search for a proper extension, albeit a drastic one, of the basic object model. Moreover, the object model which possesses strong capabilities for representing the timing behavior accurately and varying degrees of precision, is needed to support cost-effective design of real-time embedded computer systems as well. Therefore, finding such extended object models has emerged as one of the most important research issues in the real-time computing field in 1990's [Kim95a]. Ideally, a model which is capable of uniformly and accurately representing both real-time embedded computer systems and application environments, is the most desirable.

We have established one candidate for such a desirable real-time object model, called the RTO.k object model, which is also called the *time-triggered real-time object* (TT-RTO) model. An initial abstract framework of the model was formulated several years ago jointly by the first co-author and Hermann Kopetz at Technical University of Vienna. This initial framework has evolved into a concrete syntactic structure associated with unambiguous execution semantics in recent years [Kim94a, Kim94b]. The RTO.k object model was partially validated through two specification, design, and real-time simulation experiments conducted recently, one based on a defense application scenario and the other based on an advanced freeway traffic control scenario. These experiments reinforced our belief that the RTO.k model had the necessary representational power and also offered an efficient and rigorous way to develop complex real-time systems and their application environment simulators.

The RTO.k object based approach to real-time simulation has many attractive features, e.g., broad applicability, expandability and modifiability, adaptability for efficient parallel processing, etc. Real-time simulation of complex simulation targets, e.g., microscopic simulation of metropolitan area automobile traffic, cannot be realized without efficient use of distributed and/or parallel processing capabilities. The RTO.k object structuring scheme supports efficient distributed and/or parallel processing.

Execution of RTO.k objects (or any other real-time extensions of the basic object model) requires a new type of execution engines which are more reliable and predictable in meeting the timing requirements than existing widely used operating systems (including those called real-time operating systems) are. Such an execution engine must possess guaranteed timely service capabilities. Otherwise, timely actions of RTO.k objects cannot be guaranteed.

Recently an execution engine model called the DREAM (Distributed Real-time Ever Available Microcomputing) kernel was defined and then a series of its prototypes, of which the latest was called the DREAM kernel v.D3, was implemented to run on a network of PC's connected by Ethernet [Kim95b, Kim96]. It actually supports conventional processes and shared data monitors as well and thus it contains full features of a general-purpose kernel. The DREAM kernel was used in the two major experiments mentioned above.

In this paper, basic principles of the RTO.k object based real-time simulation scheme are presented. Illustrations are made by using examples drawn from the two application areas, the simulation of a defense environment and the simulation of an automobile traffic. In Section 2, the definition as well as essential properties of real-time simulation are discussed. After an overview of the RTO.k object structuring scheme is given in Section 3, the essence of an RTO.k object based approach to real-time simulation is discussed in Section 4. Section 5 provides a brief discussion on the experiments conducted recently and the paper concludes with the discussion on future research issues in Section 6.

## 2. Definition, essential properties, and applications of real-time simulation

The real-time simulation was already defined in the introductory section as a mode of simulation in which the simulation objects show the same timing behavior that the simulation targets do. Real-time simulation involves the use of a real-time clock.

The useful role of real-time simulation during development of complex real-time control computer system was already mentioned in the preceding section. After a control computer system has been implemented, it is often highly useful to connect it to a simulator of the application environment for validation of the control computer system rather than directly proceeding to interface the computer system with the application environment. A highly desirable simulator here is one capable of accurately imitating the timing behavior of the environment, i.e., real-time simulation of the environment.

Also, before a complex control computer system is fully implemented, it is often useful to do real-time simulation of the control computer system to be implemented. Obviously, such a simulator will not

contain all the detailed control algorithms needed in a fully implemented control computer system but instead perform approximate or even dummy control computations and take certain actions at the times at which a fully implemented control computer system would take the corresponding actions.

So, when we simulate physical objects in the environment, computation objects performing real-time simulation of the physical objects must show the exact timing behavior of the physical objects. The same is true for simulating control computer systems.

On the other hand, when we simulate both the application environment and a control computer system (to be implemented) together, we may choose to scale up or down uniformly in the time dimension both the environment simulator and the computer system simulator, if it serves useful purposes.

Another growing application field of the real-time simulation is the virtual reality field. A virtual reality environment corresponding to a dynamic physical environment, e.g., a compartment in a moving train or a flying airplane, is not of high quality if the virtual environment does not change at the same tempo at which the physical counterpart changes.

In a real-time simulator, the *simulator clock* must "tick" at a steady rate. Each tick of the simulator clock is commenced and administered by referencing a real-time clock in the *simulation execution engine* (a computer running the simulation program). The ticking rate of the simulator clock in a real-time 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. 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.

On the other hand, if the states of all (say, 10000) automobiles were to be updated every 100 milli-seconds for the sake of realizing high-accuracy real-time simulation, then the simulation model may impose excessive computational load on the execution engine. Therefore, the ticking interval of the simulator clock cannot be made indefinitely small.

If the simulation activities to take place during a ticking interval of the simulator clock always require a portion of the uni-processor engine power available, executing the real-time simulation model is not a difficult problem. The problem which is often non-trivial is to formulate such a real-time simulation model which contains an appropriate ticking rate of the simulator clock while realizing sufficient-accuracy real-time simulation. The probability of a low-load situation mentioned above arising becomes higher as the ticking interval of the simulator clock becomes larger, which in turn means that the simulation model becomes more crude. Suppose a large-scale simulation model is given. As the ticking interval becomes smaller, the probability of not completing all relevant simulation activities within a ticking interval becomes higher and thus the necessity of using a multi-processor (or parallel and/or distributed computer) execution engine increases. In a multi-processor execution engine, the clocks used in different processor-nodes must all be tightly synchronized.

### 3. RTO.k object structuring scheme

#### 3.1 The RTO.k object model

Only a brief overview is given in this section. More details can be found in [Kim94a, Kim96].

The basic structure of an RTO.k object is depicted in Figure 1. It is an extension of the conventional object model(s) and two most important and unique extensions are the following:

(a) Two clearly separated groups of methods:

For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at design time and such methods are called time-triggered (TT-) methods, also called the spontaneous methods (SpM's), and clearly separated from the conventional service methods (SvM's) triggered by messages from clients. The two types of methods in an RTO.k object are different not only in the way their executions are triggered but also in that

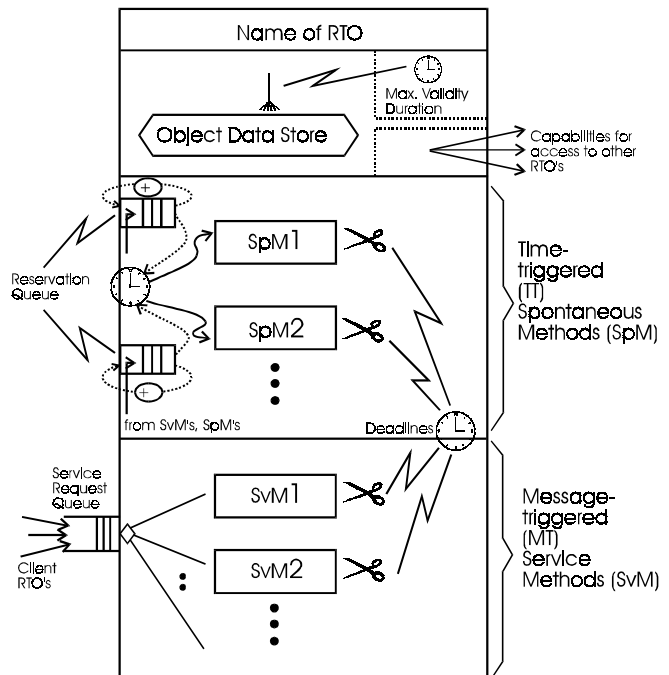


Figure 1. Structure of the RTO.k object model (Adapted from [Kim94a])

“actions to be taken at real times which can be determined at the design time can appear only in SpM's”.

Therefore, actions of the type “at constant-clock-value do S” or the type “sleep-until constant-clock-value” can appear only in SpM's. SpM's bring in new potential for concurrent executions of object methods in addition to the potential for concurrent executions of SvM's that exist in conventional objects:

- (Type I) Concurrency among SpM executions: E.g., two SpM's designed to be triggered at 10 am.
- (Type II) Concurrency between SpM executions and SvM executions.

(b) Basic concurrency constraint (BCC):

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of RTO.k objects, the execution rule which prevents conflicts between SpM's and SvM's is incorporated. 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*. To be exact, when a message-triggered SvM is not free of conflict with an SpM in accessing the same portion of the object data space (ODS), execution of the former method (SvM) must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM). This restriction is called the basic concurrency constraint (BCC). Therefore, SpM's are given higher priorities for execution over the SvM's. Note that this BCC does not impose any restriction on concurrent execution of SpM's or concurrent execution of SvM's. Therefore, executions of

SpM's are not disturbed by SvM executions and triggering times of SpM's are fixed at the design time. If a statement of the type "at 10am do S" appears in a SpM, its timely execution can be easily assured.

The above two features make the RTO.k object model clearly distinguished from other proposed real-time object models. In addition, the RTO.k object contains the following features not found in the conventional object model(s):

(c) For each execution of a method of an RTO.k object, a deadline is imposed;

(d) Real-time data contained in an RTO.k object become invalid after the interval called the maximum validity duration passes.

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  
start\_time+10min",  
"start-during (10:30am, 10:35am) finish-by  
start\_time+10min" }.

An underlying design philosophy of the RTO.k object model is that an RTCS will always take the form of a network of RTO.k objects. RTO.k objects 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. In order to facilitate highly concurrent operations of client and server objects, non-blocking (sometimes called asynchronous) types of calls (i.e., service requests) can be made to SvM's.

The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the deadline for every output produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. Before determining the deadline 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 to always execute the SvM such that the output action is performed within the deadline. Again, the BCC contributes to major reduction of these burdens imposed on the designer.

### 3.2 DREAM kernel as an execution engine for RTO.k objects

When the DREAM kernel executes RTO.k objects, it actually executes equivalent programs composed of *processes*, shared data structure monitors called *CREW* (*concurrent-read exclusive-write*) monitors, and logical

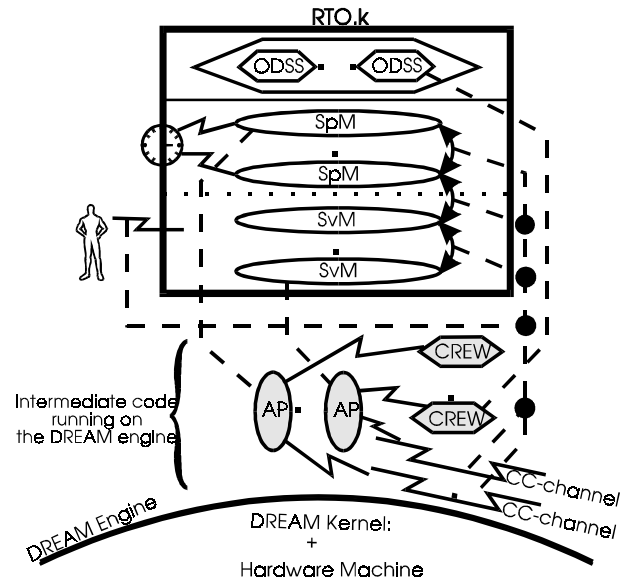


Figure 2. Mapping an RTO.k object to a process-structured-program (Adapted from [Kim95b])

multicast channels called *CC- (content code) channels*. Figure 2 depicts the mapping relationship between components of an RTO.k object and the corresponding components of an equivalent process-structured program. As shown,

- (1) object methods, both SpM's and SvM's, are mapped to processes,
  - (2) *object data space (ODS) segments* (ODSS's) to CREW monitors,
  - (3) access paths to SvM's to CC-channels, and
  - (4) result-return paths to clients to CC-channels.
- This way of utilizing CC-channels facilitates the transparency of object locations.

When an SpM is mapped to a process, the process must present to the DREAM kernel (1) the AAC of the SpM, (2) the associated deadline specification, and (3) access rights of the SpM for ODSS's. Similarly, when an SvM is mapped to a process, the process must present to the DREAM kernel (1) the associated deadline specification, (2) access rights of the SvM for ODSS's, and (3) other information related to its pipelined execution possibilities.

As mentioned earlier, several prototypes, including the up-to-date version called the DREAM kernel v.D3, was implemented to run on a network of PC's connected by Ethernet. It contains full features of a general-purpose kernel.

### 4. An RTO.k object model based approach to real-time simulation

The object model was initially formulated and used in simulation applications [Dah72]. Therefore, use of the object model in modeling the environment objects, i.e.,

modular entities in the environment which have time-varying internal states, has been practiced for a long time. However, as mentioned in the preceding section, the RTO.k object model, supports more accurate detailed modeling of environment objects.

#### 4.1 Basic approach

Suppose the application environment chosen is a sky+land+sea segment of interest, called the “theater”, and includes moving objects such as ships and airplanes, etc. This environment can be represented and simulated by the RTO.k object in Figure 3.

The object data space (ODS) in this RTO.k object contains state representations of the airplanes, the ships, and the theater space.

Each TT-method, when executed, updates a variable-set in the ODS representing the state of some physical object (i.e., airplane, ship) to reflect the current state of the physical object. 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, which may be viewed as *the ticking rate of the physical object simulator clock*, to be a fraction of the ticking rate of the real-time clock in the execution engine. Each execution of a TT-method must be completed within one ticking interval of the physical object simulator clock. Therefore, TT-methods are the mechanisms for approximately simulating continuous state changes that occur naturally in the environment objects.

The natural parallelism that exists among the environment objects is precisely represented by use of multiple TT-methods which may be activated simultaneously. In general, the accuracy of an RTO.k object structured simulation of the environment is a direct

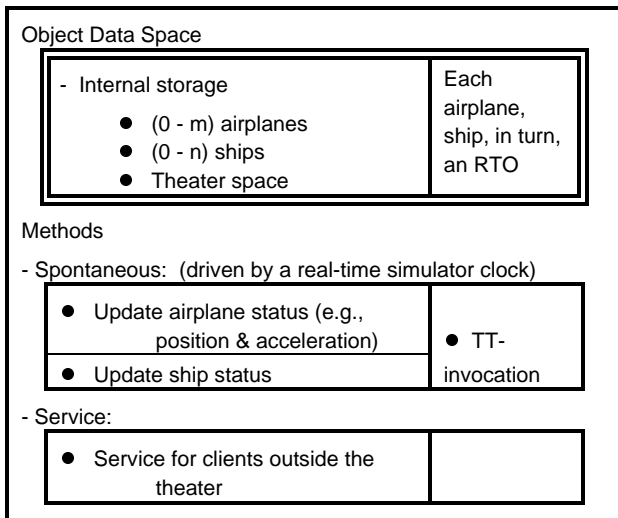


Figure 3. An RTO.k structured simulation model for a theater (Adapted from [Kim94b])

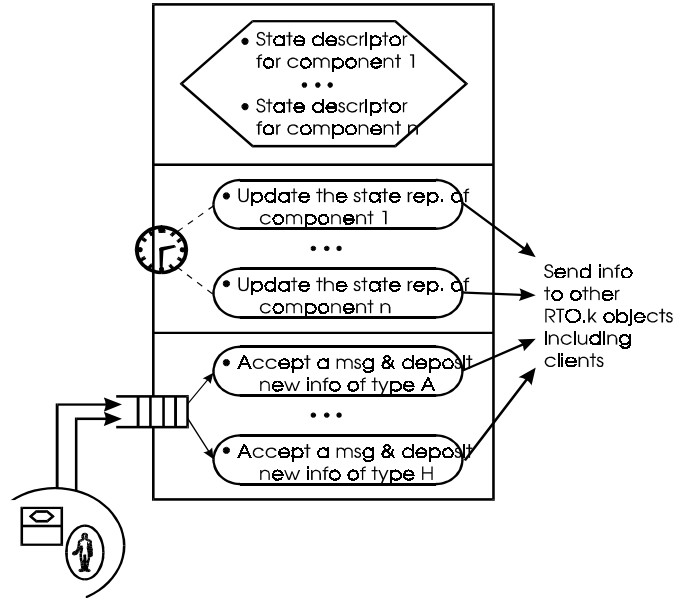


Figure 4. The general structure for the RTO.k object based simulation model function of the activation frequencies of TT-methods (which are equivalent to the ticking rates of the physical object simulator clocks).

Figure 4 depicts the RTO.k based real-time simulation approach in a generic form.

#### 4.2 Group server and monitor (GSM) object

The single RTO.k representation in Figure 3 can be expanded into a more detailed representation structured in the form of a network of RTO.k objects, each representing

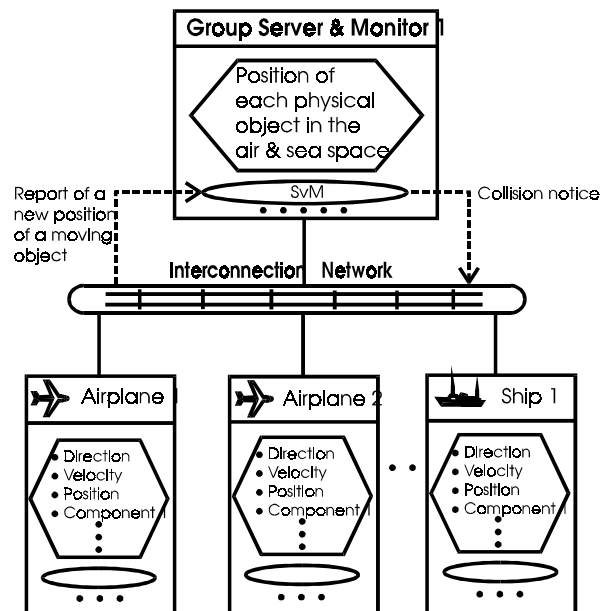


Figure 5. Decomposition of the theater RTO.k in Figure 3.

an airplane or ship, as shown in Figure 5.

Figure 5 also depicts an interesting role played by the RTO.k object representing the air and sea space. This RTO.k object maintains information on how the space is occupied. It facilitates detecting collisions between moving objects such as airplanes, etc. As the RTO.k object structured simulator of an airplane progresses after each simulator clock tick, the position of the airplane is updated and recorded within the RTO.k object. In addition, this RTO.k object notifies the RTO.k object representing the space. The latter RTO.k object in turn checks if the airplane has now collided with any other environment object such as an airplane, ship, etc., and, if so, notifies the RTO.k objects simulating the collided environment objects. The notified RTO.k objects then simulate the post-collision behavior of their simulation targets starting with the following tick of the simulator clock.

In a sense, the RTO.k object representing the space supports the RTO.k objects simulating the moving environment objects. Therefore, it is called a group server and monitor (GSM) RTO.k object. Each ticking interval of the simulator clock must cover the time spent in interaction between a GSM object(s) and monitored RTO.k objects. If the workload for a GSM object becomes too large, then multiple GSM objects, each supporting a different group of RTO.k objects simulating the physical environment objects, can be utilized.

Figure 6 depicts another example of an RTO.k object structured simulator. Automobile traffic in the Los Angeles (LA) - Orange County (OC) area is simulated. The LA district is modeled as one RTO.k object and the OC district is modeled as another. Inside each RTO.k object, dynamically changing states of cars can be traced by one or more SpM's. If the LA area simulator RTO.k object contains three SpM's which are responsible for updating the states of cars in three different segments of the LA area, respectively, and if the RTO.k object runs on a multi-processor execution engine, the three SpM's can be concurrently executed on three different processors.

Cars may move from one district to another. This is simulated by a call of the SpM for the SvM of the

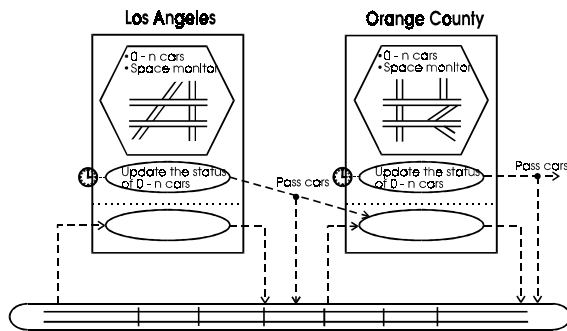


Figure 6. Simulation of cars moving from one district RTO.k to another district RTO.k

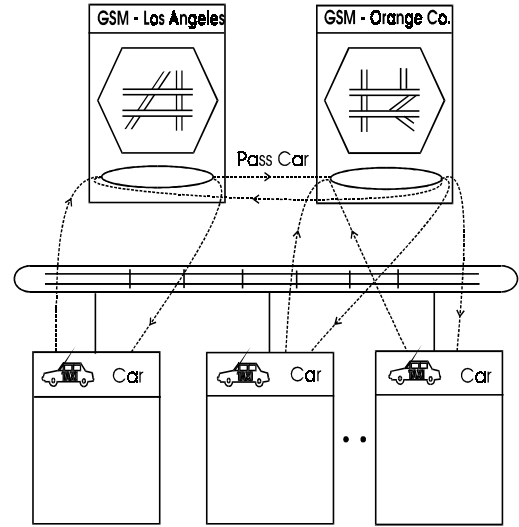


Figure 7. Car RTO's moving from one district to another district

destination district RTO which will in turn record the state of a newly entering car in the ODS.

Figure 7 depicts a variation of the simulator in Figure 6. Each car is now represented as a separate RTO. Two GSM's track space usage of the two districts, respectively.

### 4.3 Assessment

As discussed earlier, the RTO.k object model is an effective mechanism not only for variable-degree abstraction of real-time distributed computer systems (DCS's) under design but also for variable-accuracy simulation of the application environments. Structures and notations used will be of the same kind in both control system design and environment simulation. The combination of a control system design and an environment simulator takes the form of a network of RTO.k objects. Therefore, the RTO.k object structuring scheme facilitates uniform structuring of both control computer systems and application environment simulators. This presents considerable potential benefits to the system engineers.

Besides the uniform structuring capabilities, the RTO.k object based real-time simulation approach has some other unique advantages. First, due to the strong modularity characteristics of the RTO.k object, changes in the simulation target can be easily handled while achieving real-time simulation. For example, appearance of a new type of an environment object entails just adding a new RTO.k object without requiring modifications of the other existing RTO.k objects and the real-time simulation capability remains intact. Similarly, changes in some timing or logical behavior of an environment object will cause only some changes inside corresponding RTO.k

objects and again the real-time simulation capability is maintained just with that.

Secondly, in establishing a large-scale simulator, distributed and parallel (D&P) execution of simulation objects is often necessary. Such execution techniques are almost essential tools for realizing full potential benefits of real-time simulation of complex application environments or system designs. Conventional D&P simulation approaches involve message communication among the simulator execution nodes for the purpose of keeping them synchronized [Fuj90, Mis86].

In one type of D&P approaches, the processing nodes progress in parallel in a lock-step mode. They all advance to the next simulation step after ensuring that all nodes have completed the current simulation step. In large-scale real-time simulation, this message exchange overhead can be substantial since every node must generate a completion report message. It leads to a large ticking interval of the simulator clock, which in turn leads to inefficient real-time simulation.

In another type of D&P approaches, they advance somewhat asynchronously but as they often discover after exchanging messages that they have executed some incorrect simulation steps, they roll back to undo those steps. These rollback approaches also causes the ticking interval of the simulator clock to become very large, which may result in highly inefficient real-time simulation.

In contrast, no synchronization messages need to be exchanged in D&P execution of RTO.k simulator objects. This is because each simulation step with an RTO.k simulator object is triggered by a real-time clock. Therefore, as long as the triggering interval in each SpM (i.e., the inverse of the state updating frequency) in an RTO.k simulator object is chosen to be sufficiently long to cover the execution of a simulation step by any processing node, the nodes can safely start the next step simultaneously without exchanging synchronization messages. This can be more efficient in large-scale simulations which tax the power of a large-scale D&P processing system.

Thirdly, as mentioned earlier, when we simulate both the application environment and a computer system design together, we may choose to scale up or down uniformly in the time dimension both the environment simulator and the computer system simulator, depending upon the power of the simulation execution engine available. For example, by the simple action of reducing the activation frequency of each SpM without changing the body of the SpM, we can reduce the load presented to the overloaded execution engine. Such down-scaled real-time simulation, which requires a lower-power execution engine, can still serve most of the originally intended objectives of closed loop real-time simulators since the order of events and the ratio of the duration of one activity over that of another are still accurately exhibited.

## 5. Experiments conducted

As mentioned in the introduction, the RTO.k object model was partially validated through two specification, design, and real-time simulation experiments conducted recently:

- (1) An experiment that involved an application of the RTO.k structuring scheme to both the development of a defense system and that of an environment simulator and
- (2) An experiment that involved an application of the RTO.k structuring scheme to both the development of a freeway traffic control system and that of a freeway traffic simulator.

In the experiment based on the defense application scenario, the environment considered was a much more complicated version of the theater shown in Figure 3. In this environment, there were hostile flying objects, radars, and interceptors (both on the land and on ships) in addition to the airplanes and ships. Control computer systems were used both on the land and on ships.

In the experiment based on the advanced freeway traffic control scenario, electro-magnetic loop detectors (which detect cars moving over the loops) and ramp meters were considered in addition to freeway lanes. Car accidents were also dealt with. Control computer systems were used to collect the loop detector data and control ramp meters under heavy traffic conditions.

In both experiments, each entire application environment was treated as a single RTO.k object initially. Then the single RTO.k object specification was gradually refined into a network of RTO.k object specifications, each corresponding to a different environment object or a computing object. RTO.k objects were then distributed among multiple PC's running the DREAM kernel and exchanging messages over an Ethernet. Some PC's were dedicated to graphic display functions.

These experiments reinforced our belief that the RTO.k model had the necessary representational power and also offered an efficient and rigorous way to develop complex real-time systems and their application environment simulators. Although the experiments were conducted on a local area network of PC's rather than on a sizable parallel computer, the benefits mentioned in Section 4.3 could still be confirmed by and large. Some real-time fault tolerance techniques were also incorporated into these experimental control computer systems.

## 6. Future directions

The real-time simulation can play useful roles in many phases of the complex system engineering cycle, in particular, validation and evaluation of control computer system designs as they evolve through abstract designs to detailed implementations. Not only logical behavior but also timely performance and dependability characteristics

can be effectively validated and evaluated via real-time simulation.

The RTO.k object based approach to real-time simulation has been presented in this paper. The RTO.k object is capable of uniformly and accurately representing both real-time embedded computer systems and application environments. The RTO.k object based approach to real-time simulation has many attractive features, e.g., broad applicability, expandability and modifiability, adaptability for efficient parallel processing, etc. Much further work is needed to develop supporting tools for the RTO.k object based real-time simulation. First of all, a high-level language tool for RTO.k object structuring which is more abstract in nature than the language C++ combined with an RTO.k object component library, e.g., the DREAM library in [Kim96], needs to be established along with a supporting translator. We are currently developing an extension of C++, which we have named C++T. Secondly, an efficient execution engine, especially one capable of effectively taking advantage of the raw processing power of highly parallel machines, needs to be developed. Adapting the DREAM kernel to a highly parallel machine such as Intel Corp.'s Paragon, is considered to be a worthwhile research and development effort.

One important feature that is essential for a real-time simulation system is visualization. Ideally, real-time display of the dynamically changing states of the simulation targets should not fall behind the real-time simulation beyond a certain fixed time interval. Visualization of RTO.k objects is thus another meaningful topic for future research and development.

**Acknowledgment:** The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the California Transportation Department via the UCI ITS, in part by the University of California MICRO Program under Grant No. 93-080, in part by Hitachi, Ltd, and in part by Postech.

## References

- [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.
- [Kim94a] 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.
- [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 and Applications Conf. (COMPSAC)*, Nov. 1994, Taipei, pp.392-402.
- [Kim95a] Kim, K.H., "Toward New-Generation Real-Time Object-Oriented Computing", *Proc. IEEE CS 5th Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, Cheju Island, Aug. '95, pp.520-529.
- [Kim95b] Kim, K.H. et al., "A Timeliness-Guaranteed Kernel Model -- DREAM Kernel -- and Implementation Techniques", *Proc. RTCSA '95 (1995 Int'l Workshop on Real-Time Computing Systems & Applications)*, Tokyo, Oct. '95, pp.80-87.
- [Kim96] Kim, K.H., Subbaraman, C., and Kim, Y.S., "The DREAM Library Support for PCD and RTO.k programming in C++", *Proc. 1996 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Feb. '96, Laguna Beach, pp.59-68.
- [Mis86] Misra, J., "Distributed Discrete-Event Simulation", *ACM Computing Surveys*, Vol. 18, No. 1, March 1986, pp.39-65.
- [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.