□ 研 究 論 文 □

韓 國 軍 事 運 營 分 析 學 會 誌

第 25 卷, 第 1 號, 1999. 6. 30

Event-Driven Real-Time Simulation Based On The RM Scheduling and Lock-free Shared Objects

Park, Hyun Kyoo*

Abstract

The Constructive Battle Simulation Model is very important to the recent military training for the substitution of the field training. However, real battlefield systems operate under real-time conditions, they are inherently distributed, concurrent and dynamic. In order to reflect these properties by the computer-based simulation systems which represent real world processes, we have been developing constructive simulation model for several years. Conventionally, scheduling and resource allocation activities which have timing constraints are major problem of real-time computing systems. To overcome these constraints, we elaborated on these issues and developed the simulation system on commercially available hardware and operating system with lock-free resource allocation scheme and rate monotonic scheduling.

* BCTP

1. Introduction

The Army's C2(Command & Control) training simulation models are interactive computer-driven systems that assist training readers, commanders, and their staffs to develop and maintain the unit readiness. This is well-known as a constructive simulation.

The constructive simulation system is one of the famous real-time system software, and the one common feature of all real-time systems is defined as the correctness of the system depend not only on the logical result of computation, but also on the time at which the results are produced.

Most work on implementing the constructive simulation model has focused on using the priority-driven scheduling and lock-based critical sections to ensure object consistency with a cost ineffectiveness, thus it specifically increased operating system overhead.

Our method is allocating computational resources to preemptible, lock-free shared objects with static priority. The formal comparison based on the previous research and the our experimental results are of interest because they avoid priority inversion and deadlock with no underlying operating system support for object sharing. Eventually, this approach to accomplish the real-time constraints is techniques to deal with the mission spaces of military simulation system. In a hard real-time system, unless the scheduling processes are carefully controlled, it may be difficult or impossible to ensure that task deadlines are always met.

2. Theoretical Background And Previous Studies

2.1 Terminologies and Notations

There are some key definitions that apply through this paper are enlisted below.

Event : A change of object attribute value, an interaction between objects, an instantiation of a new object, or a deletion of an existing object that is associated with a particular point on the simulation time axis. Each event contains a time stamp indicating when it is said to occur.

Logical Time : A simulation system's current point on the logical time axis used to specify before and after relationships among events. Specifically, the simulation requests advances in logical time via Time Advance Request and the simulation system notify the advance of logical time through the Time Advance Grant service.
Wallclock Time : a simulation system's measurement of true global time, where the measurement is typically output from a hardware clock.

2.2 An Analytical Framework

The term schedulability means the ability of a process or a set of process to meet deadlines. The analysis of the schedulability of various hard real-time systems was performed and investigated. For the real-time simulation, we assume that there are n processes on an uni-processor. And the amount of time c_i that task T_i spends performing an operation with accessing objects without nested calls is

$$c_i = u_i + m_i \cdot t_{acc} \text{ , where} \qquad \text{[Eq. 1]}$$

u_i : the computation time not involving accesses to shared objects

m_i : the number of shared objects accesses by T_i

t_{acc} : maximum computation time for any object access

In this paper, we discuss the complexity of the real-time scheduling algorithms achieving lag bounds of <1 or better. Generally, a schedule for which the number of allocations to any task is at all times within an additive Δ of the ideal value is said to achieve a lag bound of Δ.

Informally, lag of task T at slot t with schedule S is

$$Lag(S, T, t) = W(T, t) - A(S, T, t) \qquad \text{[Eq. 2]}$$

where $W(T, t)$ represents the ideal number of times for task T to be assigned the objects and $A(S, T, t)$ denotes that task T is allocated the objects under schedule S in slots 0 through t-1. A lag bound L is a pair $(<L, \Delta L)$ where $<L$ is either $<$ or $\leq$, and ΔL is a real number. For the sake of brevity, we refer to these lag bound as $(<, 1)$ and $(\leq, 1 - 1(2n - 2))$.

A scheduling algorithm achieves a lag bound of L if and only if any schedule S produced by the algorithm satisfy $| Lag(S, T, t) | < L\Delta L$ for all tasks T and slots t.

Former research, several scheduling paradigms emerge, depending on schedulability analysis, and whether the result of the analysis itself produces a schedule or plan according to which tasks are dispatched at run-time. Based on this, previous research identified by the following classes of algorithms described in <Table 1>. The evaluations and characteristics of these approaches can be found in [7].

We define the constructive simulation and the events as a problem of dynamic task set with variable task requests.

Table 1. Scheduling on task attributes

Analysis	Task Dispatch	Content
Static	Table Driven	Applicable Periodic tasks Dispatched by the table
	Priority Driven Preemptive	Static or dynamic Priority (DM, RM, EDF, etc)
Dynamic	Planning Based	Feasibility is checked at run-time
	Best Effort	Feasibility is not guaranteed Task may be aborted

2.3 Adopted Scheduling Paradigm Via Formal Comparison

Our examination of the schedulability of various event paradigms considered the static priority driven scheduling. Initially, a set of independent periodic processes, where independent means that the processes do not have synchronization requirements and periodic means the processes are initiated at regular periods and have deadlines at the end of the period. The several static preemptive scheduling approaches are described as follows.

Deadline-Monotonic (*DM*) : A static priority scheme in which tasks with shorter relative deadlines have higher priorities.

Rate-Monotonic (*RM*) : A static-priority scheme in which tasks with shorter periods have higher priorities.

Earliest-Deadline-First (*EDF*) : A dynamic-priority scheme in which, at any given instant, the task that has the closest deadline has highest priority

In the paper [1: Anderson et. al]., described that the comparing the overhead of lock-free object sharing under *RM* scheduling with the overhead of the lock-based

priority ceiling protocol(PCP). Under the PCP, the worst-case blocking time equals the time required to execute the longest critical section. From this, we denote the schedulability condition for periodic tasks using the PCP as,

$$PCP \equiv \forall i \exists t : 0 < t \leq p_i : r + \sum_{j=1}^{i} \lceil \frac{t}{p_j} \rceil (u_j + m_j \cdot r) \leq t, \text{ where} \qquad \text{[Eq. 3]}$$

$u_j + m_j \cdot r$: the computation time of task T_j.

In the above equation, the first term on the left-hand side represents the blocking factor. The expression on the left-hand side represents the maximum demand due to T_i and higher priority tasks in a interval of length t.

In the paper [3: Klein et. al] showed the non-preemptibility is a source of blocking. If consider the case where processes is not only performed in a mutually exclusive manner, but is also non-prermptible. Perhaps the process is non-preemptible because of object requirements or merely because the process was implemented in this manner. All other assumptions remain the same. The non-preemptible sections represent a source of blocking to higher priority tasks.

This consequences derived the preemptive scheduling and allocation strategy for the simulation with formerly developed constraints.

However, The lock-free based on RM scheduling approach to real-time object sharing that we investigated was done by previously some twenty years ago in the real-time systems community. But this idea was forgotten for many years.

3. Scheduling and Resource Allocation

3.1 Rate Monotonic Real-Time Scheduling

For many years, *Rate Monotonic*(RM) scheduling theory offers a set of engineering principles for managing timing complexity.

The rate monotonic algorithm assumes priority-based preemptive scheduling where a process's priority is based on its period.

To apply RM scheme in this simulation, it is assumed that task deadlines and periods coincide, i.e., each invocation of a task must complete execution before the next invocation that task begins.

Under *RM* scheme, the conditions for the schedulability of a set of periodic tasks that share lock-free objects were formerly investigated. For brevity, the proof of this condition is omitted here, and the formal proof of theorems can be found in [1].

From the previous research, the following two theorems give the necessary and sufficient conditions for the scheduling by *RM* scheme in this simulation.

Theorem 3.1. (Necessity under *RM*) If set of periodic tasks that share lock-free objects is schedulable under the *RM* scheme, then the following condition holds for every task T_i.

$$\exists t : 0 < t \leq p_i : \sum_{j=1}^{i} \lfloor \frac{t}{p_j} \rfloor \cdot c_j \leq t$$

The left-hand side of the quantified expression gives the minimum demand - which arises when there are no interferences - placed on the processor by T_i and higher-priority tasks in the interval [0, t] where $0 < t \leq p_i$. The right-hand side gives the available processor time in that interval.

Theorem 3.2. (Sufficiency under *RM*) A set of periodic tasks that share lock-free objects is schedulable under the *RM* scheme if the following condition holds for every task T_i.

$$\exists t : 0 < t \leq p_i : \sum_{j=1}^{i} \lceil \frac{t}{p_j} \rceil \cdot c_j + \sum_{j=1}^{i-1} \lceil \frac{t-1}{p_j} \rceil \cdot s \leq t$$

where

N : The number of tasks in the system

i, j : Task indices quantified over {1,...,N}

p_i : The period of task T_i, $p_i < p_j \Rightarrow i < j$

T_i : i_{th} task in the system

c_i : The computational cost of task T_i

s : The execution time required for one loop iteration in the implementation of a lock-free object, which for simplicity is assumed to be the same for all objects

3.2 Time Stamp Event Ordering

We limited dynamic event occurrence and execution by time stamp. Each event is

able to access a clock and acquire the estimate of the current logical time $F_i(t)=t \pm \Delta i(t)$, through the time service. $F_i(t)$ is the monotonic function that maps real-time to logical time and Δi is the latency by the network and operating system overhead.

Using time stamp, the events can be isolated as a task only when they do not overlap other tasks where the overlap is with respect to time or shared objects. The following definitions describe the time stamp and serialized event execution.

Definition 3.1. A time-stamp is a real number that represents the occurrence time of an instantaneous event, and is an indivisible entity.

Definitions 3.2. There are two binary relations between time stamp.

- $\tau_\alpha < \tau_\beta$ (τ_α before τ_β) and its inverse $\tau_\alpha > \tau_\beta$ (τ_α after τ_β)
- $\tau_\alpha = \tau_\beta$

Definition 3.3. An execution T is a serial execution, Serial(T), of task T *iff* :

$\forall e_i, e_j \in T : i \neq j \rightarrow$

$([\forall f(i,c_i) \in e_i, f(j, c_j) \in e_j : (f(i,c_i), f(j, c_j)) \in R] \wedge [\forall f(i,c_i) \in e_i, f(j, c_j) \in e_j : (f(j, c_j), f(i,c_i)) \in R])$.

It must be possible to order the tasks T_1, T_2,..,T_n such that T_1 accesses the initial state of database, and T_2 accesses the database that would have been produced if T_1 had run to completion.

Theorem 3.3. A property of serial executions, which is a requirement for event-driven simulation, is that each event executes upon a consistent database state. This property holds assuming the initial database state d_0 is consistent, and the event is executed in isolation, it will preserve the database consistency, and the system doesn't allow for the existence of arbitrary states after the execution of transactions.

Proof : A consistent database state d_n will be obtained from the initial, consistent database state d_0 if the events are executed serially, since there will be an ordering of the tasks T_1, T_2,..,T_n, such that T_1 sees the initial state d_0, for each $i > 1$ transactions T_i sees the consistent state, d_i, that will be produced by running T_1, T_2,..., T_n sequentially to completion, and d_n is the state produced by T_n .

In the simulation, events are delivered to *Event-Handler* in time stamp order. Thus the events will not be processed until the event with a smaller time stamp being

remained. To accomplish this approach, *Event-Handler* will hold incoming events in its queue by the time stamp and also ensure that no event is delivered to a *Event-Handler* containing a time stamp less than the system's current logical time. This eliminates certain temporal anomalies that might otherwise occur when various clients send different ordering events. And each event has only convex(contiguous) execution duration with consistent database in this simulation.

3.3 Lock-free Object Sharing

The best object allocation is assigning the object to T in times $\lfloor w \cdot t \rfloor$ or $\lceil w \cdot t \rceil$ of the first t slots for all.

Traditionally, the simulation models used lock-based objects under various scheduling. The performance of lock-free and lock-based approaches were described in [1]. In this paper, given hardware support for primitives of compare & swap, the result of value s in lock-free object approach varies from 1.3 μsecs for a counter, to 3.3 μsecs for a circular queue. In contrast, lock-based implementations fared much worse in a recent performance comparison of commercial real-time operating systems. Although this result cannot be regarded as definitive, they do give some indication as to the added overhead when operating system based locking mechanisms are used.

Formal comparison can be derived through [Eq. 3] in section 2.3. Representative lock-free operations are usually implemented using "retry loops," such as <Figure 1>.

$$\forall j : j \le i : (m_j + 1) \cdot s \le m_j \cdot r \wedge PCP \qquad \text{[Eq. 4]}$$

{substitute $(m_j + 1) \cdot s$ for $m_j \cdot r$ *in* PCP }

$$\Rightarrow \forall i \exists t : 0 < t \le p_i : \sum_{j=1}^{i} \lceil \frac{t}{p_j} \rceil (u_j + m_j \cdot s) + : \sum_{j=1}^{i} \lceil \frac{t}{p_j} \rceil \cdot s + r \le t \qquad \text{[Eq. 5]}$$

$$\Rightarrow \forall i \exists t : 0 < t \le p_i : \sum_{j=1}^{i} \lceil \frac{t}{p_j} \rceil (u_j + m_j \cdot s) + : \sum_{j=1}^{i-1} \lceil \frac{t-1}{p_j} \rceil \cdot s \le t \qquad \text{[Eq. 6]}$$

Because $c = u_j + m_j \cdot s$ in the lock-free case, the last expression in this derivation is equivalent to the scheduling condition of theorem 2. In this case, $s < r/2$ implies that

$$\forall j : j \le i : (m_j + 1) \cdot s \le m_j \cdot r$$

since, for positive m_j, $\frac{1}{2} \leq m_j / (m_j + 1) < 1$.

Thus if the time taken to execute one iteration of a lock-free retry loop is less than half the time it takes to access a lock-based object under the *PCP*, then any task set that is schedulable under the *PCP* is also schedulable when using lock-free objects. In [Eq. 4], s is the cost of one lock-free retry loop.

In Figure 1, a message is inserted in the buffer by using a *write* instruction to update the buffer status by changing a tail pointer and either the next pointer of the last item in the buffer or a head pointer, depending on whether the buffer is empty. This loop is executed repeatedly until the *write* instruction succeeds.

The buffer is not explicitly locked by any task, so it is essential property of lock-free implementations that operations may interfere with each other. An interference results in this example when a successful *write* by on task results in a failed *write* by another task. However, it is not immediately apparent that lock-free shared objects can be employed of tasks must adhere to strict timing constraints.

Figure1.Lock-freeSharedBuffer

```
class SharedBuffer
public variable
Head, Tail : pointer
AccessKey : Integer
method write (stream)
private variable
Old, New : pointer
do
    if Old != Null then add stream;
    else add stream; send signal;
until EventRemaind()
```

3.4 Guaranteed Worst-Case Response

One of the most distinguishing feature of the static priority scheduling is that it can't provide guaranteed timely services to real-time applications when the importance of task is changed. The dynamic or adaptive scheduling might resolve this problem.

When the cumulative request of all tasks of an object exceeds the capacity of the object, we extends the scheduling strategy with an amendments. It is to identify a maximal subset of the requesting tasks whose requests can all be accommodated. If all the requests made of an object of capacity C sum to R, $R > C$, each task is offered a fraction C/R of its requested capacity. Then the *Time Advance Request* will change the progress ratio adaptively to extend the capacity C.

4. Experimental Results

4.1 Modeling Feature

Regiment And Battalion Tactics Simulation were fielded by Army from 1995 to 1997, providing regiment and battalion commanders and their staff with training in combat and battlefield operations and procedures of the tactical decision establishment. These simulation exercises maneuver, fire support, air defense, engineer, tactical air, Army aviation, logistics, maintenance, personnel administration, and higher headquarters functions.

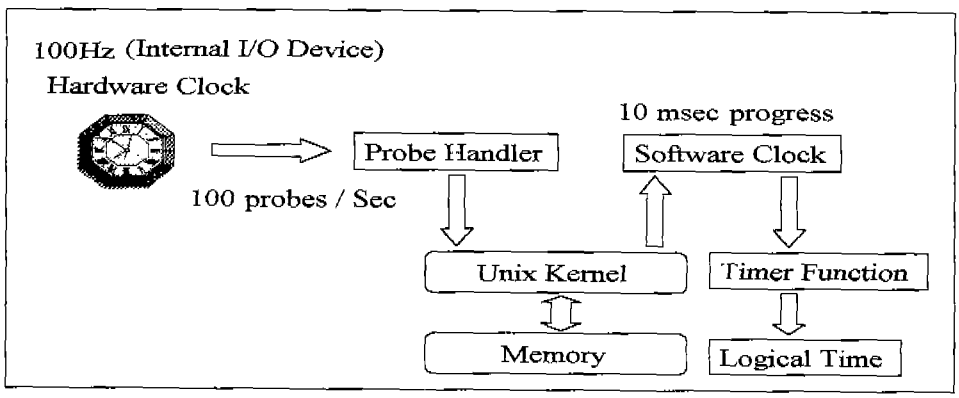

Figure 2. Logical time generation

The simulation model is constructed on single CPU Sun Ultra-workstation and IBM PCs connected via local network for clients. The clients have their own Geographical Information System(GIS) module and user interaction components to transmit the events and receive the result messages.

For the logical time in this simulation system, we use the Unix software

clock(Solaris clock) and distribute the logical time to all processes to synchronize the entire system. This time service called *Time Advance Grant(TAG)* distribute the logical time each minute by the progress ratio. Also, sometimes the progress ratio would be changed by the client request and then this task will update the scale factor of software clock either gathering the logical time or loosing time through *Time Advance Request (TAR)*.

Each event has time stamp by the logical time and ordered by its time stamp before input to the event queue.

Alert Message	Game Status Save Game Start / Restart Time Progress Ratio Change	Real-Time Channel
Real-Time Message	Orders Spot Messages Clock Message Unit Status Retrieval	
Non Real-Time Message	Report Generation Database Retrieval	Non Real-Time Channel

Table 2. Messages and channels

Lock-free scheme as described, the shared memory is widely used for the message transmission among the processes and clients which are attached via the network and controlled by *Network Daemon*.

The *Event-Handler & Game Controller* has the key role of simulation both for the logical computational result and the control of event processing (ordering, retraction, etc). The Figure 3 depicts the highest level of the main program of the simulation.

For the client, the display of the terrain and units with volatile messages is the essential component module. The Figure 4 shows the current display screen image.

The terrain is represented by a 100 meter feature with 50 meter elevation grid data consistent with the 1:50K scale topographic map. The play areas range in scale from 1:25K to 1:100K, although the usual scale is at 1:50K. This huge database is the important parameter for the precise simulation results. To display of the terrain, we used our own GIS module for the clients. However, the details of client component is beyond of this paper.

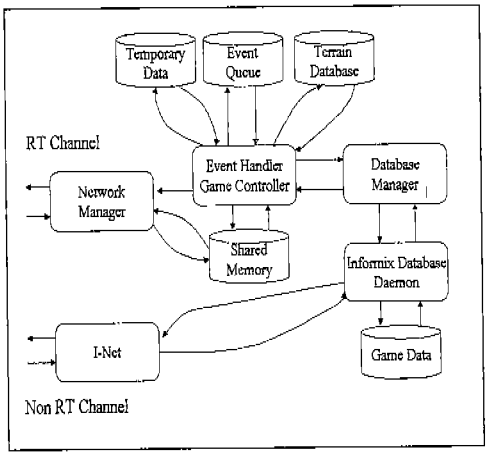

Figure 3. Top-Level Software Architecture

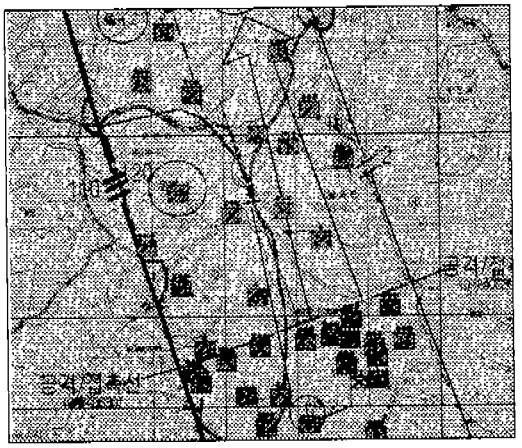

Figure 4. The display of client screen

4.2 Network Manager

In the simulation, there are three classes of messages with different requirements. We can classify three message classes as following.

Alert messages are aperiodic and have strict delivery time constraints. They are considered to represent critical conditions, hence they are allowed to interfere with normal task executions. Messages in this class requires the guaranteed transmission routes to guard against transmission errors and components failures.

Real-time messages, also have time-constraints on their delivery time. These include both periodic and aperiodic messages. An important subclass of this are the *Clock Messages* which are exchanged between processes as part of a clock synchronization procedures. The arrival pattern for aperiodic messages can be specified using a pessimistic estimate for the minimum inter-arrival time. We assume that an occasional loss of messages, due to transmission errors or because of a time-constraint violation caused by the presence of *Alert Messages*, is permissible for real-time messages. This may occur in the real battlefield by the unexpectable communication failure.

Messages of the third class do not have hard time-constraints, so their scheduling and routing is much more flexible. This class also includes messages without any expressed timing constraints, like those found in general purpose client-server

systems. The system has separate transmission channel for this purpose. It is only required to provide "best-effort" delivery for these messages.

The communication system has to support all three types of messages. In <Figure 3>, there are two message channels. One for first and second class messages and the other is third class messages. In these three types, real-time messages pose many problems because they require delivery time guarantees. Hence, we designed special communication Network Manager(*NM*) which is oriented toward real-time messages, and *I-Net daemon* has the responsibility of the third class messages.

NM maintains state information for all clients in the system and implements the channel establishment scheme outlined in Figure 3. The *NM* maintains several structures which are vital to the channel establishment procedure. A table containing the *Client ID* and the resource requirements for all existing channels in the system.

For the non real-time messages, channel establishment is a local operation. The parameters of the channel are stored in a session structure, so that they can be used during the actual message transmission. The mechanism used for the transmission of these messages is described in [2].

4.3 Database Manager

Databases are partial models that provides a means to record facts about particular aspects of the simulation. Implicitly, database have state and at every instant the database state is the collection of all the facets it contains.

However, just as a model changes to remain current, so must a database change. Changes are introduced into a database by the execution of *Datanase Daemon*(*DD*). *DD* is a program that access and manipulate the Informix Daemon for the database transactions. As changes occur, the transaction system insures the preservation of the database's consistency. But, in this paper, we omit the data gathering and report generating component which is important part of the client operation.

As described in section 4.2, the system is consisted with a separate data gathering and reporting communication channel, which trigger the database engine directly. This means that the simulation system has unpredictable long-term events. This infrastructure constitutes the basis for aggregation of simulation results and reporting

of the running system under real application load.

4.4 Current Status

Various problems exist when attempting to convert a non real-time operating system to a real-time version. These problems can exist both at the system interface as well as in the implementation. In particular, the POSIX P.1003.4 subcommittee is defining standards for real-time operating system. To date, the effort has focused on eleven important real-time related functions; timers, priority scheduling, shared memory, real-time files, semaphores, interprocess communication, asynchronous event notification, process memory locking, asynchronous IO, synchronous IO, and threads.

However, we developed the real-time simulation on commercially available operating system. In the near future, a common approach which is to encapsulate software scheduling algorithms into a fast but general purpose operating system, called a Real-Time Operating System (RTOS) will solve many problems of scheduling and allocation. The basic idea is to provide the functionality needed by real-time software systems without the large overhead associated with traditional operating systems. A good overview of RTOS research in scheduling algorithms is contained in [1][7].

The GIS module with VPF format digital maps is currently under development.

5. Concluding Remarks

Scheduling real-time computations is an extremely important part of a real-time system because it is the phase in which we assign the actual temporal properties to the computations. The time management structure of this simulation system is intended to support interoperability among heterogeneous platforms utilizing different internal time management mechanisms. Thus the logical time of the simulation is precisely guaranteed not only for the master game controller but also for the clients. Eventually this execution support distributed time advance mechanism for the next generation distributed simulation model in [9]. And our preliminary results proved that the lock-free shared object with static priority scheduling is affordable for the

real-time simulation on uni-processor system. These efforts will remove special hardware and the problem-oriented real-time operating system support for the wargame area. However, this scheme needs further experiment to decide if it can be effectively applied to the hard real time simulation.

If the real-time kernel contains a scheduler with a priority scheduling algorithm, a dispatcher which controls the task switch mechanism, a wait queue for inactive tasks, a wait queue for tasks waiting for a time event, and a ready queue described in this paper, then the simulation system will exclude additional effort of developing game control mechanisms.

The formal comparisons and progresses of the scheduling method is the ongoing job in this area. Our objective is to provide and support an abstraction which allows expression of the hardware and software requirements of real-time applications. Specifically, we consider the issue of guaranteeing the delivery of message objects with time constraints with the correct computational results presently.

참고문헌

[1] Anderson, J. et. al, "Real-Time Computing with Lock-Free Shared Objects", Proceedings of the 16th IEEE Real-Time Systems Symposium, Dec. 1995.

[2] Dean, T., Greenwald, L., Kaelbling, L., "Time Critical Planning and Scheduling Research at Brown Univ.", Tech. Report CS-94-41, 1994..

[3] Klein, M., Ralya, T. "An Analysis of Input/Output Paradigms for Real-Time Systems," CMU-SEI-PO-TR-19, Carnegie Mellon Univ. 1990.

[4] Levi, S. and Agrawala, " Real-time System Design", McGrow-Hill, 1990.

[5] Massalin, H., "Synthesis: An Efficient Implementation of Fundamental Operating System Services," Columbia Univ. 1992.

[6] Rajukmar, R., Synchronization In Real-Time Systems - A priority Inheritance Approach. Kluwer Academic Pub. 1991.

[7] Ramamritham, K. and Stankovic, J., "Scheduling Algorithms and Operating Systems Support for Real-Time Systems," CS Technical Report, Univ. of Messachsetts, 1996.

[8] 박현규 외, "TMO모델을 이용한 분산 실시간 워게임 시뮬레이션 모델의 개발", Proceedings of KSS 춘계학술대회, 1998.

[9] 박현규, "RM 스케줄링과 Lock-Free 공유개체에 의한 실시간 시뮬레이션", Proceedings of IEEK 추계학술대회, 1998.

[10] High Level Architecture Rules v. 1.3 DMSO, Feb. 1998.

[11] High Level Architecture Time Management Design Document v.1.0 DMSO,. 1996

[12] Informix Reference Guide, Informix Inc. 1997.

[13] Training with Simulations, National Simulation Center, Nov. 1996.