Timing Issues and Experiment Scheduling in Faster-than-Real-Time Simulation

Dimosthenis Anagnostopoulos
Harokopio University of Athens
70 El. Venizelou Str, 17671
Athens, Greece
email: dimosthe@hua.gr

Mara Nikolaidou
University of Athens
Panepistimiopolis, 15771
Athens, Greece
email: mara@di.uoa.gr

Abstract

Faster-than-real-time simulation (FRTS) can be used for the performance evaluation of systems behavior in real time, providing significant capabilities for studying systems with a time-varying behavior. FRTS enables model validation through comparing simulation results with the corresponding system observations. However, experimentation proves to be rather demanding, as both delivering output results and ensuring their reliability must be accomplished within a predetermined time frame. Output analysis of system observations and model results and relevant timing issues are discussed. A method is introduced for determining the "optimal" faster-than-real-time experiment, in terms of the number of replications that may be scheduled for execution, as well as whether it is required to make a compromise between the ability to predict for the long future and the degree of reliability achieved for predictions. Experimental results are presented to validate and demonstrate the effectiveness of the proposed method. A case study where this method is applied for reaching FRTS predictions, contributing to efficiently servicing client requests in a central library information system, is also discussed.

1. Introduction

A real-time simulator is a real-time system where some portions of the environment, or even portions of the real-time system itself, are realized by a simulation model [1]. Evaluating the performance of a system under study in real time is certainly not a trivial task, depending on various issues, such as the system speed (i.e. how often state changes occur, compared to world time) and nature, which determines crucial issues, such as the allowed degree of human interaction with the system. When attempting to reach conclusions for the system behavior in the near future, faster-than-real-time simulation (FRTS) is widely used. In this type of simulation, advancement of simulation time occurs faster than real world time.

Real time systems often have hard requirements for interacting with the human operator or other agents [2]. Making models run faster is the modeler's responsibility. When being handled at the implementation level, timing problems are recognized during or even after testing, and this may cause considerable inefficiencies. Researchers have thus pointed out that timing requirements should be addressed at the design phase [3]. Addressing timing problems at this phase, though, does not consider the variability in the time required to execute an experiment, which is caused by real conditions, such as arrival-process distribution parameters, which may be non-stationary. It does not also consider the need for achieving a specific degree of confidence for simulation results, which may only be calculated during the execution of the experiment. We thus argue that timing issues should also be addressed at the implementation (or execution) phase. Regarding model validation in RTS, in which domain-oriented approaches are usually employed, e.g. for validating control system models or computer network models [4], we consider that it should also be addressed at the methodological level through generic approaches.

Timing requirements such as the following are considered in FRTS: the model must run faster than the system, system data must be obtained and processed in real time so that the current system state is always known, the model has to be adapted to the current system state without terminating the FRTS experiment, as the system under study may be a time-varying one, and dynamic (i.e. in real time) model modification should be enabled. In addition, model validity must always be ensured and, when simulation results are utilized to ensure validity, they must have essential statistical properties (e.g. the appropriate sample size and statistical processing). Analysis of simulation results and system data must always be performed within the given time frames. Timing requirements can be addressed by methodological approaches, such as [5], which is employed in this paper. This conceptual FRTS methodology provides extensions to traditional simulation activities, focusing on real-time interaction and ensuring the validity of simulation results. It consists of the following phases: *modeling, experimentation and remodeling*. The modified FRTS executive is presented in figure 1. Both the system and the model are under monitoring during experimentation. Data depicting their consequent states are obtained within predetermined time intervals of equal length, called *auditing intervals*. In case the model state deviates from the corresponding system state, remodeling is invoked. This may occur due to system modifications, which can involve its input data, operation parameters and structure [5]. In these cases, remodeling adapts the model to the current system state. This is accomplished without terminating the real time experiment, since no recompilation is performed. When model modifications are completed, experimentation resumes. Remodeling can also be invoked when deviations (expressed through appropriate performance measures) occur between the system and the model due to the stochastic nature of simulation, even when the system parameters/ components have not been modified. In case simulation results (predictions for the near future) are considered to be valid, an additional phase, called *plan scheduling,* is invoked to take advantage of them [5]. However, as depicted in figure 1, plan scheduling activities are not considered as a part of the FRTS environment.

Experimentation encompasses *monitoring*, which is the activity where system and model states are obtained and stored while the model is executed, and *auditing*, which accomplishes model validation tasks (figure 1). Auditing examines if simulation results are reliable and if the system has been modified during the last observation, based on system observations. The dynamic system behavior may result in critical modifications in the system input data, operation parameters and structure. Structure variability, in particular, has been studied either at the methodological level [6], [7] or in the context of domain-oriented research approaches. To conclude about such modifications, specific attributes of both systems are put under monitoring. The corresponding variables are referred as *monitoring variables.* Monitoring variables should be considered at a conceptual level, not according to the conventional, single-valued definition of program variables. Auditing examines the monitoring variables corresponding to the same real time points (i.e. the current system state and simulation predictions for this time point) and concludes for the evolution of the system and the model [5].

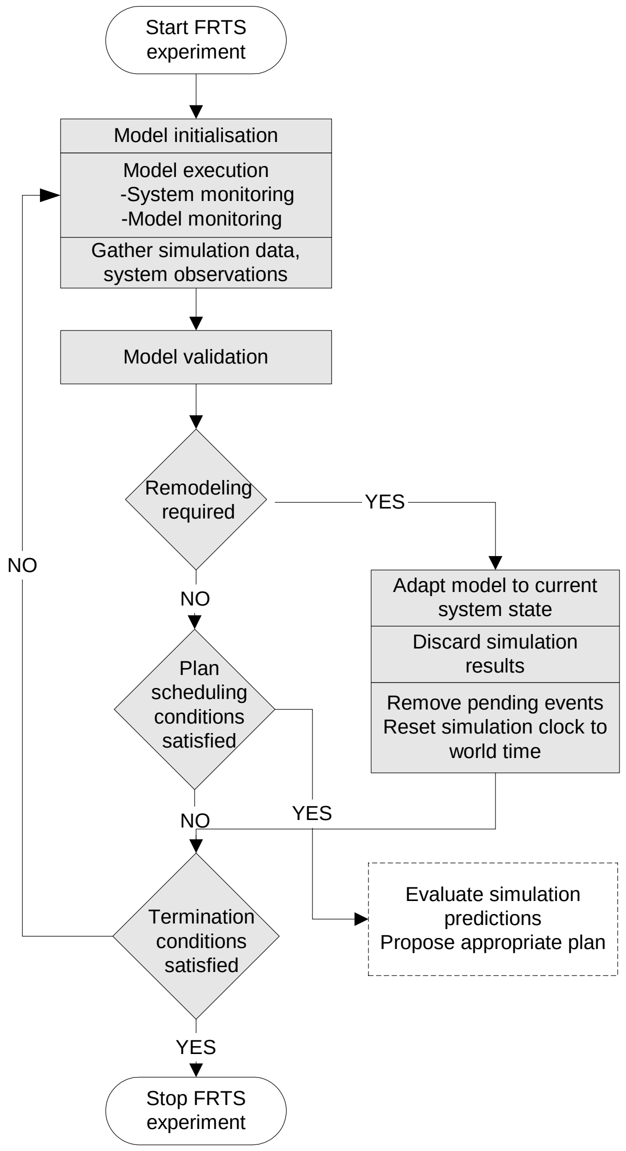
Start FRTS experiment
Model initialisation
Model execution -System monitoring -Model monitoring
Gather simulation data, system observations
Model validation
Remodeling required
YES
NO
Adapt model to current system state
Discard simulation results
Remove pending events Reset simulation clock to world time
NO
Plan scheduling conditions satisfied
NO
YES
Evaluate simulation predictions Propose appropriate plan
Termination conditions satisfied
YES
Stop FRTS experiment

Figure 1: FRTS executive

An example is presented in figure 2, where the evolution of the system and the model is depicted at the two horizontal axes. Real time points are noted as t_i. The states of the system and the model at point t_i are noted as R_i and S_i, respectively. When at t_x the model predicts the system state at t_n (simulation time is equal to t_n) we use the notation *Sim*(t_x)= t_n. States S_x and R_n are thus compared during auditing at time point t_n. The auditing interval length (e.g. [t_{n-1}, t_n], [t_n, t_{n+1}]) remains constant throughout the experiment, and is noted as *AudInt.*

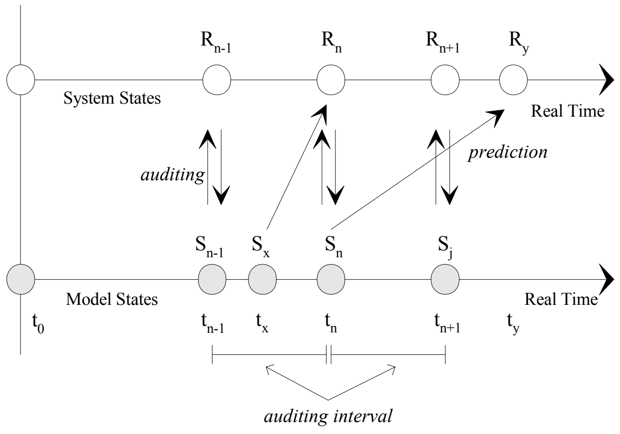

Figure 2: Experimentation in FRTS

In this paper, we address timing issues of FRTS, that is, we state the problems encountered and propose a method for executing experiments conforming to the real-time requirements. The proposed experiment scheduling method is independent of the execution environment, as it applies to both sequential and parallel execution. However, in the latter case, processing units must be of the same hardware and have the same load, as latter discussed. The term FRTS (or FRTS experiment) denotes the entire simulation process, involving model execution, monitoring, auditing and remodeling activities, performed in consecutive auditing intervals. FRTS experiments are terminated when termination conditions are fulfilled (e.g. if FRTS cannot be achieved or world time reaches a predetermined value). As, in the general case, simulation does not reach a steady state within each auditing interval, a single experiment performed within an auditing interval involves multiple replications. This imposes that multiple replications must be completed within the given time frame to reach predictions. Towards this objective, a discussion concerning timing issues in FRTS resides in section 2. A method for scheduling experiments and dealing with timing requirements is presented in section 3. Section 4 includes experimental results in order to examine the validity and efficiency of the proposed method. Section 5 presents a case study where the proposed method is applied for reaching FRTS predictions, contributing to efficiently servicing client requests in a central library information system. Finally, conclusions reside in section 5.

2. Timing issues

Each FRTS experiment is executed in consecutive auditing intervals. Within each interval, an autonomous subexperiment is executed, aiming at reaching results for the future states that must be predicted. To achieve reliability, each subexperiment involves *n* independent replications, each one terminated when simulation time reaches a specific future time point. Timing issues thus concern accomplishing *n* replications within a given time frame. A description of how FRTS experiments are executed as terminating simulations involving multiple replications has been given in [8].

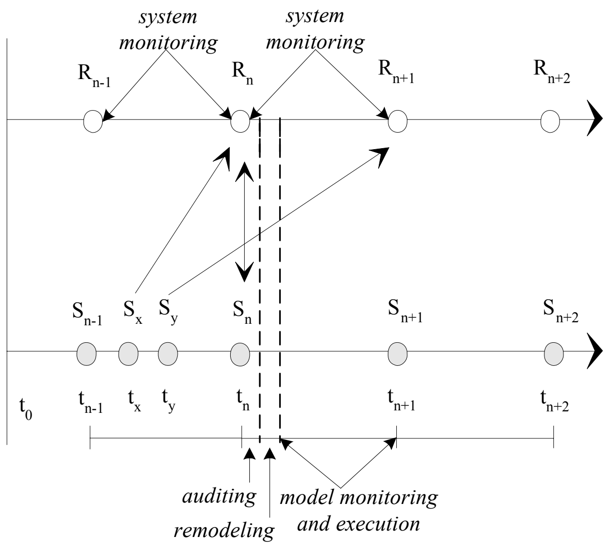

Figure 3: Execution of monitoring, auditing and remodeling within the auditing interval

We use the auditing interval (*AudInt*) as a time unit to specify the future time interval for which we wish to reach predictions, denoted as the *prediction interval*. Then, *prediction interval* = $p*AudInt$, $p \in N^*$. Suppose that simulation starts at point t_{n-1}, as depicted in figure 3. To achieve FRTS, results for p intervals ahead of t_{n-1} must be reached within interval $[t_{n-1}, t_n]$. As $Sim(t_x) = t_n$, and $Sim(t_y) = t_{n+1}$, model results are obtained at the respective real time points t_x and t_y. Note that t_x and t_y are the time points where all replications are completed for the corresponding predicted intervals, e.g. at t_x, the "slowest" replication produces its output to form R_n.

Except from the first interval (i.e. $[t_{n-1}, t_x]$), where the initial model state has to be identical to the corresponding system state, in all other cases, the model state at the end point of the previous interval serves as the initial state for the next one. Thus, there are *p* sets of output results produced within a single auditing interval and *p* terminating events exist in each "complete" (i.e. covering all predicted intervals) replication. The terminating event examines which are the time points (e.g. t_j) at which simulation reaches results for each one of the forthcoming *p* intervals, so that:

$$Sim(t_j) = t_{n-1+i} = t_{n-1} + i*AudInt,\ i \in N^*,\ 1 \leq i \leq p$$

The above apply on the basis that a fixed number of replications is required to reach predictions for each of the future system states (i.e. R_n, R_{n+1}, …), which is rather a common practice. In this way, reliability is ensured with the same amount of experimental results for each of the predicted states.

Determining the prediction interval length (i.e. *predictability*) is dictated by the requirements imposed to simulation (e.g. for a control processes) and by the constraints imposed by the nature of the system (e.g. how fast the system evolves). Reaching reliable predictions for long ahead is also a desirable feature. On the other hand, reliability tends to decrease for long-ahead predictions, as the characteristics of real systems tend to change over time. In addition, timing restrictions are increased, especially when the system evolves nearly as fast as simulation does. Supposing that simulation reaches conclusions for at least *p* auditing intervals ahead of real time, if current time is t_{n-1} and $Sim(t_{n-1}) = t_{n-1}$ (i.e. simulation is re-initiated), then, at some future point t_j, the following must apply:

$$Sim(t_j) = t_{n-1+p} = t_{n-1} + p*AudInt$$

$$t_j < t_n = t_{n-1} + AudInt$$

To discuss analytically timing issues, we examine the FRTS activities that need to be accomplished within the given time frame. The real-time data flow diagram [9] of figure 4 presents the control process and the time-consuming activities (in gray) of FRTS, which are: *model initialization*, *system* and *model monitoring*, *auditing* and *remodeling*. The process that accepts experiment parameters from the user is accomplished prior to the initiation of the FRTS experiment, thus not consuming real time. System monitoring is executed continuously and concurrently with model execution during the auditing interval. Thus, timing requirements involve model initialization and model monitoring (i.e. model execution and storing output results), consuming time equal to T_{Init} and T_{Exec}, which must be completed before the upper endpoint of the current auditing interval.

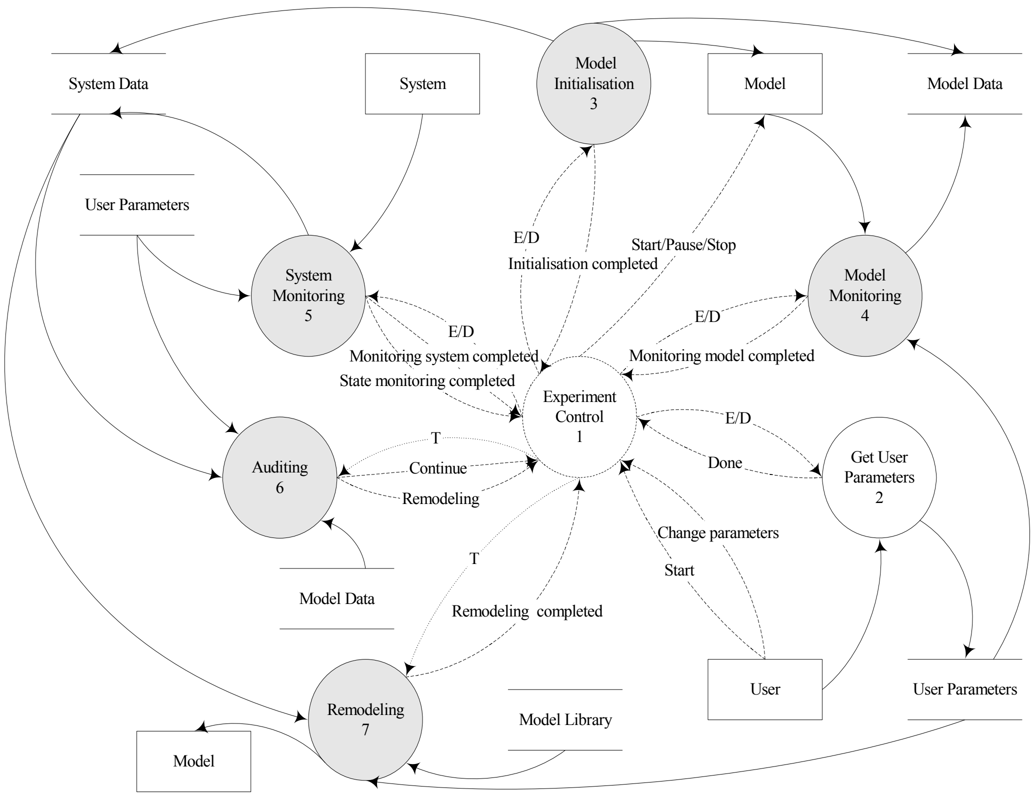

Figure 4: Real-Time Data Flow Diagram of FRTS control

Auditing is rather different, since it is executed only when the auditing interval has elapsed. However, there are strong timing requirements for the time it consumes, as model execution is temporarily paused while model validity is examined. Auditing must thus be accomplished with minimum time overhead before resuming experimentation. The same requirements apply for remodeling, which -when needs to be executed- is invoked after auditing. Remodeling additionally discards all previous simulation results and the FRTS experiment is re-initiated from the current real time point.

In figure 3, $Sim(t_x) = t_n$ and $Sim(t_y) = t_{n+1}$. Auditing thus compares S_x with R_n at t_n. Assuming that auditing indicates that the corresponding model and system states do not coincide, predictions (i.e. S_y) are discarded and remodeling is activated to restore consistency. The time consumed is equal to T_{Audit} and $T_{Remodel}$, respectively. We note that:

1. System monitoring process is re-initiated after the completion of the previous auditing interval without waiting for the completion of auditing and monitoring activities. In this way, we avoid ignoring significant system changes during the execution of these activities and we ensure that auditing is always performed at the expected time point (i.e. $t_{n-1} + i*AudInt$).
2. Model monitoring/execution is actually performed during a shorter time frame than *AudInt* (i.e. reduced by T_{Audit} and $T_{Remodel}$). The duration of both auditing and remodeling activities should thus be taken into consideration when determining the available time frame for model execution. However, T_{Audit} and $T_{Remodel}$ shorten the time frame for executing the subexperiment of the forthcoming interval, not the time frame of the current auditing interval (i.e. T_{Audit} required for data of $[t_{n-1}, t_n]$ reduces execution time in $[t_n, t_{n+1}]$).

To finally formulate the timing requirements for executing simulation within the given time frame, we consider the general case, also the most demanding, as follows:

1. Both auditing and remodeling have been performed, consuming time equal to T_{Audit} and $T_{Remodel}$. Simulation time must thus be re-initiated from the starting point of the current auditing interval.
2. Simulation must reach predictions for *p* auditing intervals ahead of this point.

The essential condition for accomplishing FRTS within each auditing interval is thus:

$$T_{Audit} + T_{Remodel} + T_{Init} + T_{Exec} \leq AudInt \qquad (1)$$

In (1), T_{Init} is practically equal to 0. T_{Exec} is the time required to reach predictions for *p* intervals ahead. As T_{Audit} and $T_{Remodel}$ concern the previous auditing interval, these values are already available when a new experiment is initiated within the current interval.

3. Experiment scheduling method

Suppose that T_i is the time required for the sequential execution of replication *i* out of *n* independent replications that must be performed. We define $b_i = T_i \,/\, PredictedInterval$ to express how much slower is the system than simulation in replication *i*. Then,

$$T_{Exec} = \sum_{i=1}^{n} T_i = \sum_{i=1}^{n} b_i (pAudInt) = pAudInt \sum_{i=1}^{n} b_i \quad (2)$$

Combining (1) and (2),

$$T_{Audit} + T_{Remodel} + pAudInt \sum_{i=1}^{n} b_i \le AudInt$$

$$\sum_{i=1}^{n} b_i \le \frac{AudInt - T_{Audit} - T_{Remodel}}{pAudInt} \quad (3)$$

Condition (3) is the one determining if it is possible to perform FRTS on a software and hardware platform offering the capability to run each replication *$1/E(b_i)$* times faster than the system (*$E(b_i)$* is the expected value of *b_i*). Based on this, the following method is proposed for executing multiple replications. The method combines theoretical and experimental results to examine whether it is possible to execute *n* replications, each one producing results for *p* auditing intervals ahead of the starting point, within the given time frame. A number of initial replications (n_o) is used to estimate $\mu=E(b_i)$.

Step 1: Make n_o replications of the simulation and gather statistics (note that n_o should be large enough to provide a good estimation for μ).

Step 2: Estimate μ and $\sigma^2=Var(b_i)$ as follows: $\mu = \frac{1}{n_0}\sum_{i=1}^{n_o} b_i$, $s^2 = \frac{1}{n_0 - 1}\sum_{i=1}^{n_o}[b_i - E(b_i)]^2$.

Step 3: Suppose that $B_n = \sum_{i=1}^{n} b_i$. Assume there is distribution function for b_i. We know that b_1, b_2, … b_n are IID (Independent and Identically Distributed), thus s^2 is an estimate of σ^2. Using the central limit theorem, $(B_n - n\mu)/\sigma\sqrt{n} \sim N(0,1)$. Then, $(B_n - n\mu)/s\sqrt{n} \sim N(0,1)$. We want the probability that *n* replications are executed within the given time to be equal to *a* (e.g. 0.9). Thus,

$$P[B_n \le \frac{AudInt - T_{Audit} - T_{Remodel}}{pAudInt}] \ge a.$$

Suppose that $\lambda = \frac{AudInt - T_{Audit} - T_{Remodel}}{pAudInt}$. Then,

$$P[B_n \le \lambda] = P[(B_n - n\mu)/s\sqrt{n} \le (\lambda - n\mu)/s\sqrt{n}] = P[Z \le (\lambda - n\mu)/s\sqrt{n}] \ge a.$$

Calculate k so that $P[Z\leq k] = a$. Thus, to perform n replications within the time frame given with probability a, the following condition must be fulfilled:

$$(\lambda - n\mu)/s\sqrt{n} \geq k \qquad (4)$$

Step 4: If condition (4) can be fulfilled, calculate n and execute the rest n-n_o replications. When results from all replication are produced for each interval, update statistics (using also the results from the n_o replications) and store results. If the time consumed during simulation exceeds *AudInt*, immediately terminate simulation.

Step 5: If condition (4) cannot be fulfilled, a decision has to be made concerning decreasing the number of remaining replications (n-n_o) so that $(\lambda - n\mu)/s\sqrt{n} \geq k$ or decreasing p. After that, execute the remaining replications. When results from all replications are produced for each interval, update statistics (using also the results from the n_o replications) and store results. If the time consumed exceeds *AudInt*, immediately terminate simulation.

Using this method, we cannot be certain that results from all replications will always be produced within the given time frame, due to the stochastic nature of simulation. In any experiment that simulation exceeds the given time, the specific experiment is immediately terminated. To ensure that results for the most immediate intervals are produced with an acceptable degree of reliability, which depends on the number of replications performed, concurrent execution of replications may be employed. In this way, reliability of results for the most immediate intervals, for which we are mostly concerned as they are soon to be compared with system data, is not endangered. If, for instance, some replications were not to produce results within the give time frame, there would be an impact on reliability and remodeling could be caused. However, this must be avoided, since all predictions would have to be discarded and remodeling time would have to be consumed. After all, if predictions for "remote" intervals cannot be obtained, they may as well be produced within the next auditing interval. The authors argue that this principle should also be adopted when considering whether to decrease the number of remaining replications or the number of predicted intervals in step 5. Assuming that a minimum number of replications n_{min} must be performed to ensure reliability, p must fulfill condition:

$$p \leq \frac{AudInt - T_{Audit} - T_{Remodel}}{AudInt\,(ks\sqrt{n_{min}} + n_{min} * \mu)} \tag{5}$$

Based on the above, the proposed scheme for executing *n* independent replications is depicted in figure 5. The initial n_o replications are executed sequentially. The remaining n-n_o ones may be executed concurrently, so that all replications first produce their output for t_{n+1}, then for t_{n+2} and last for t_{n+p}.

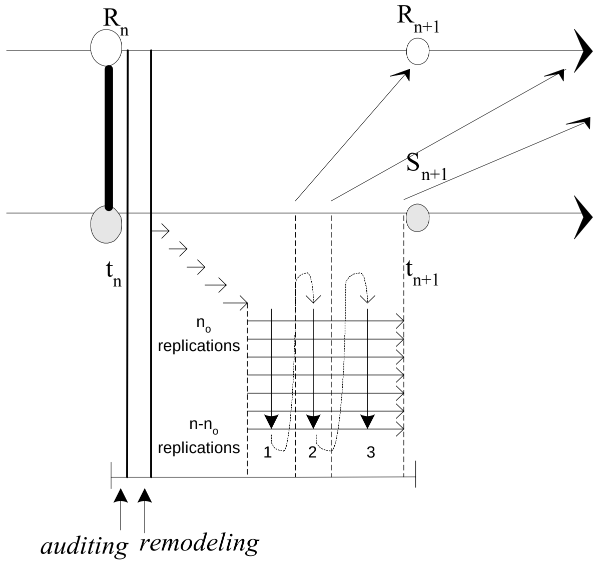

Figure 5: Proposed scheme for executing *n* replications

The proposed experiment scheduling method is independent of the execution environment, as it applies to both sequential and parallel execution. Parallel simulation is most suitable for executing the remaining n-n_o replications, to ensure that results for the most immediate intervals are produced with an acceptable degree of reliability (step 5). However, processing units must be of the same hardware and have the same load, in order to run each replication $1/E(b_i)$ times faster than the system. Furthermore, when executing independent, sequential simulations on different processors (also known as the *replicated trials* approach), a drawback is that each processor must contain sufficient memory to hold the entire simulation [10]. In the case of large-scale systems, T_{Audit} and $T_{Remodel}$ could be considerably increased, but this should not be a problem as they refer to the previous auditing interval and are thus known when a new experiment is initiated in the current interval. A potential problem could emerge due to unreliable communications, especially in a LAN or WAN computing environment, preventing the delivery of some replication results. Reaching results on the basis of the rest $n_o–1$ or *n-1 replications* (provided that these are available) could be an acceptable solution.

Applying the proposed method also requires that monitoring capabilities be provided over T_i, that is, the time required to execute replication *i*. Calculating T_i is rather trivial and, as numerous output variates are calculated within each replication, T_i can be considered as an additional output variate, which does not impose any limitation to this method.

4. Experimental validation

To test the correctness and efficiency of the proposed method, we performed numerous experiments using standard queuing models, with the following two objectives: a. to verify that *n* experiments can actually be accomplished within the given time frame with probability *a*, where *n* is determined at runtime, and b. to estimate the improved reliability of simulation results, after the execution of the maximum number of potential replications. To solely explore the above critical issues, we separated the execution of replications from any particular real system. Conducting experiments without real-system interaction, irrelevant factors, such the duration of T_{Audit}, $T_{Remodel}$ and the availability of system monitoring data when initiating auditing, did not interfere with the validation process.

In each experiment, we executed n_o initial replications, calculated the number of potential replications *n*, and then executed the remaining n-n_o ones, to examine if all replications were completed within the given time frame. We considered a probability of 95% for being able to execute *n* replications within the auditing interval ($a= 0.95$). Each experiment was repeated 100 times to determine the percentage of successful experiments and compare it against *a*. The average delay in the queue (*avgD*) is the output variate of each replication.

To conduct experiments, we modeled the following types of systems, which are widely used in the literature:

QNM1: a M/Gamma/s queuing network (using Kendall's notation in classical queuing theory), that is, a multiple-server/single-queue system with *Exponential(b)* interarrival times, *Gamma(c,β)* service times and *s* servers, for s=*64, 128, 256*, various values of *b* and *c*, and $\beta = 2.0$.

QNM2: a queuing network with *s* stations (multiple-server/multiple-queue system) where jobs are randomly routed to stations, with *Exponential(b)* interarrival times, *Gamma(c,β)* service times, for *s=64, 128, 256*, various values of *b* and *c*, and *β = 2.0*.

Both sequential and distributed FRTS experiments were performed. In sequential simulation, the execution environment was a Sun Ultra 5 with 1 CPU and 640 Mbyte running Solaris 8. In distributed simulation, the environment consisted of 5 Sun Ultra 5 with 1 CPU and 128 Mbyte running Solaris 8, under conditions of equal load. Auditing and prediction intervals are equal to 5.0 sec/15.0 sec and 10.0 sec/30.0 sec, respectively, so that predictions had to be reached for 3 auditing intervals ahead (*p=3*). For each combination of interarrival and service parameters *b, c* and the number of servers *s*, distinct experiments were conducted according to the following algorithm:

- Make 100 repetitions of steps 1-5 for either experiment type (sequential, distributed)		
	Sequential	*Distributed*
1	Execute n_o replications and calculate statistics	Execute at least n_o replications, equally distributed among processors, and calculate statistics
2	Calculate n so that condition (3) is fulfilled with probability *a*	
3	Execute the remaining *n*-n_o replications and calculate statistics	Execute at least *n*-n_o equally distributed replications and calculate statistics
4	Determine if the experiment is successful (i.e. if the time required to execute *n* replications is less than the auditing interval)	
5	Calculate the precision increase (*pri*) in results after *n* replications compared to the corresponding precision after n_o replications (*pri*)	
- Calculate the percentage of successful experiments (*srate*) and the average precision increase *avg(pri)* from all 100 repetitions		

After each completion of n_o and *n*-n_o replications, we build a *(1-l)%* confidence interval for *avgD*. In our case, *l=10%,* so that a *90%* confidence interval is built. We use the average value of the confidence-interval half-length $\delta(n,l)=t_{n-1,1-\frac{l}{2}}\sqrt{\frac{S^2(n)}{n}}$ divided by the point estimate $\overline{avgD}(n)$ as a measure of the precision of the confidence interval [11]. Results for sequential and distributed simulation successful experiments, for various combinations of interarrival times and service times, are given in table 1 for NQM1 and in table 2 for NQM2. Parameters *(s,b,c)* of each experiment are depicted in the rows marked with gray. For NQM1, results from 27 different combinations are presented, also for NQM2. The average number of experiments where replications were executed within the given time frame is noted as *avg(srate)*. In most cases, the results obtained are far better than expected, i.e. *avg(srate)>α*, indicating that experiments were accomplished as scheduled.

S	interarrival and service times (b, c)								
64	0.01, 0.608	0.01, 0.624	0.01, 0.640	0.02, 1.216	0.02, 1.248	0.02, 1.280	0.03, 1.824	0.03, 1.872	0.03, 1.92
seq.	0.973	0.956	0.99	0.95	0.953	0.986	0.99	0.966	0.993
distr.	0.98	0.966	0.976	0.966	0.97	0.956	0.976	0.953	0.96
128	0.01, 1.216	0.01, 1.248	0.01, 1.280	0.02, 2.432	0.02, 2.496	0.02, 2.56	0.03, 3.648	0.03, 3.744	0.03, 3.84
seq.	0.986	0.983	0.986	0.996	0.973	0.986	0.99	0.983	0.97
distr.	0.966	0.99	0.98	0.976	0.966	0.97	0.98	0.966	0.973
256	0.01, 2.432	0.01, 2.432	0.01, 2.496	0.02, 4.864	0.02, 4.992	0.02, 5.12	0.03, 7.296	0.03, 3.488	0.03, 7.68
seq.	0.98	0.99	0.956	0.98	0.976	0.993	0.966	0.95	0.973
distr.	0.967	0.98	0.96	0.986	0.95	0.99	0.953	0.96	0.966

Table 1: *avg(srate)* in sequential and distributed simulation (NQM1)

s	interarrival and service times (b, c)								
64	0.01, 0.512	0.01, 0.576	0.01, 0.640	0.02, 1.024	0.02, 1.152	0.02, 1.28	0.03, 1.536	0.03, 1.728	0.03, 1.92
seq.	0.99	0.996	0.97	0.986	0.97	0.966	0.98	0.993	0.986
distr.	0.973	0.973	0.99	0.963	0.963	0.976	0.963	0.976	0.97
128	0.01, 1.024	0.01, 1.152	0.01, 1.28	0.02, 2.048	0.02, 2.304	0.02, 2.56	0.03, 3.072	0.03, 3.456	0.03, 3.84
seq.	0.99	0.976	0.993	0.98	0.956	0.97	0.963	0.976	0.963
distr.	0.966	0.96	0.99	0.973	0.96	0.99	0.99	0.976	0.98
256	0.01, 2.948	0.01, 2.304	0.01, 2.56	0.02, 4.096	0.02, 4.608	0.02, 5.12	0.03, 6.144	0.03, 6.192	0.03, 7.68
seq.	0.99	0.986	0.976	0.993	0.99	0.98	0.963	0.966	0.956
distr.	0.973	0.953	0.99	0.963	0.956	0.97	0.953	0.97	0.95

Table 2: *avg(srate)* in sequential and distributed simulation (NQM2)

Executing the maximum number of potential replications, the average increase in the precision of simulation results is depicted in table 3. The precision obtained after n_o and n replications is equal to $\delta(n_o,l)/\overline{avgD}(n_0)$ and $\delta(n,l)/\overline{avgD}(n)$, respectively. Precision is increased when the average value of the confidence-interval half-length $\delta(n,l)= t_{n-1,1-\frac{l}{2}}\sqrt{\frac{S^2(n)}{n}}$ divided by the point estimate $\overline{avgD}(n)$, after the execution of n replications, is less than the respective value obtained after n_o replications, that is, when $pri<1$, where

$$pri = (\delta(n,l)/\overline{avgD}(n))/(\delta(n_o,l)/\overline{avgD}(n_0))$$

Precision increase results with distributed simulation are presented in table 3 for NQM1. As *avg(pri)* ranges from 0.616 to 0.846, an increase of 15.4% up to 38.4% is obtained, contributing significantly to the reliability of predictions.

s	interarrival and service times (b, c)								
64	0.01, 0.608	0.01, 0.624	0.01, 0.640	0.02, 1.216	0.02, 1.248	0.02, 1.280	0.03, 1.824	0.03, 1.872	0.03, 1.92
avg(pri)	0.713	0.720	0.706	0.663	0.658	0.671	0.616	0.616	0.622
128	0.01, 1.216	0.01, 1.248	0.01, 1.280	0.02, 2.432	0.02, 2.496	0.02, 2.56	0.03, 3.648	0.03, 3.744	0.03, 3.84
avg(pri)	0.789	0.768	0.773	0.684	0.692	0.702	0.644	0.647	0.656
256	0.01, 2.432	0.01, 2.432	0.01, 2.496	0.02, 4.864	0.02, 4.992	0.02, 5.12	0.03, 7.296	0.03, 3.488	0.03, 7.68
avg(pri)	0.820	0.846	0.843	0.697	0.702	0.741	0.648	0.673	0.674

Table 3: Average precision increase after the execution of n-n_o replications (NQM1)

5. A case study

We present a case study where the proposed scheduling method was employed in the FRTS system supporting handling client request for a central library information system. We discuss applicability and efficiency issues, as being capable of executing *n* replication with the given time frame and improving the precision of results were substantiated in the previous section. Search requests for the University of Athens Library database server are issued through the web from more than 20,000 academic users (students and staff). Searching is also enabled for external users. During morning peak hours, more than 100 concurrent users may issue bibliographical search requests for the central database. The library uses the Horizon library automation software [12] for managing bibliographical data, stored in a Sybase RDBMS. Approximately 500,000 bibliographical records are maintained. The Sybase DB server maintaining Horizon databases runs in a Unix cluster, composed from two IBM F50 machines, each equipped with 2 processors and 1Gb RAM, with mutual takeover capabilities. Requests are issued through the web and are serviced as follows (figure 6).

1. End users issue requests from client workstations.
2. Requests are transferred through HTTP to the WebPac (i.e. the standard web interface of Horizon) server and then transmitted to the DB server using a Z39.50 client. Z39.50 is the worldwide standard bibliographical data exchange protocol [13].

3. Requests are received from the Z39.50 server and then serviced by the DB server. Results are then emitted back to end users in reverse order.

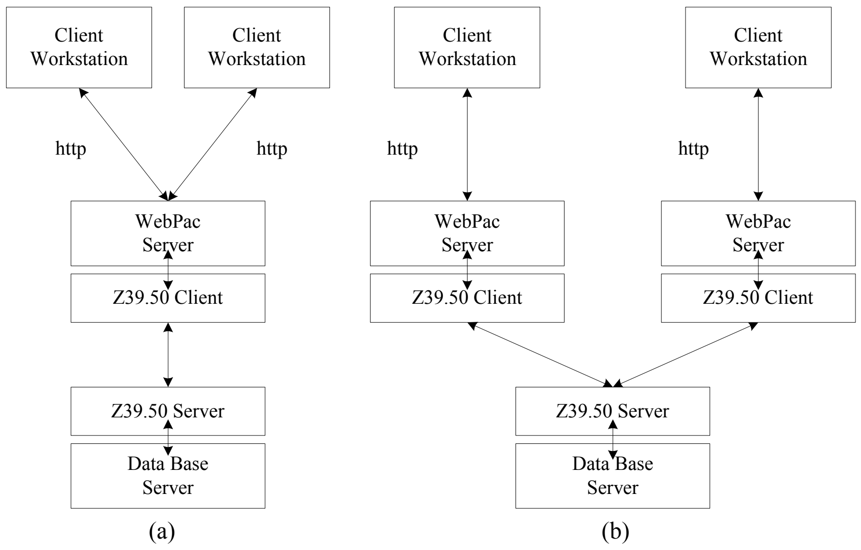

Figure 6: Search request servicing (a) normal operation (b) using additional WebPac servers

In the architecture depicted in figure 6, WebPac servers run only on MS Windows NT or MS Windows 2000 platforms and are multithreaded. Z39.50 clients are not. A single Z39.50 client instance may run on each machine for transmitting search requests to the DB server. It was often observed that the performance of the WebPac server was not efficient during morning peak hours, as responses to search requests delayed for more than 5 min before results were presented to end users. A thorough examination indicated that this was due to the processing requirements of WebPac, which consumed a significant time overhead for presenting search results to end users (all results are presented as web pages). On the other hand, the performance of the DB server was never problematic, i.e. the time required to process requests and send them back to the Z39.50 client was constantly less than a maximum value (≈ 2.5 sec).

A potential solution to this problem was to initiate more than one instance of WebPac/Z39.50 on separate servers. This is depicted in figure 6 (b). Applying this solution, a considerable downsizing (more than 50%) in the delay was achieved. However, handling peaks like this imposed that additional servers, such as CD-ROM servers, be used for running WebPac/Z39.50 processes, in addition to their original tasks. As this may not be employed in a mandatory basis, FRTS techniques were used to indicate whenever utilizing additional servers was essential for handling peaking search requests.

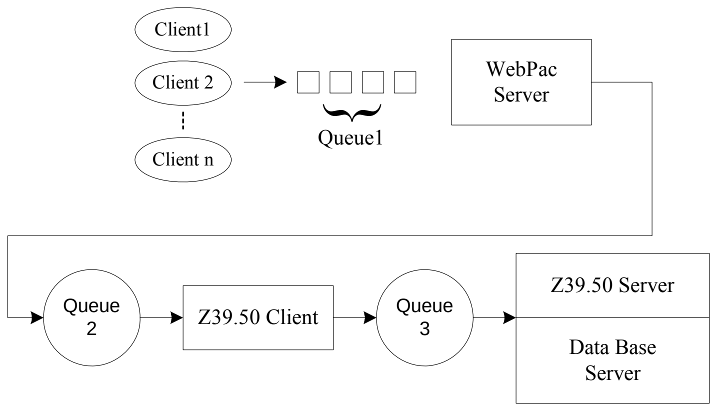

Figure 7: Request processing simulation model

The simulation model for accepting and processing search requests is depicted in figure 7 (*Queue1* is different to depict multithreading). The FRT simulator runs on an Intel-based 850 MHz workstation with 256 Mbytes of RAM. The monitoring variables used for model validation are the number of requests (*reqno*) in the current interval and the average response time (*avgreponse)*. The following parameters were used for simulation experimentation: $AudInt=60\ sec$, $p=10$, $n_o=30$. The simulator runs concurrently with the information system, using its current data concerning the requests being issued and serviced. Predictions are reached for $p=10$ intervals ahead of world time, so that alternative servers can be activated before critical delays are encountered. Predictions are considered as valid when found to be reliable in at least 4 consecutive invocations of auditing. In this way, additional servers are used only when needed, not just when counters exceed a predetermined threshold, as this may be circumstantial.

In figure 8, the (increasing) number of search requests in consecutive auditing intervals is depicted for three different cases (data have been shifted so that they start from point 0.0). Model predictions are also depicted, both the valid ones used to activate additional servers as well as invalid predictions, which are marked with dotted lines. Invalid predictions are discarded when remodeling is performed; new predictions are then produced at later time points. The points where predictions are ultimately considered as valid are explicitly noted with a larger marker. The points where remodeling is invoked are depicted on the lower graph for each of the three cases, using 0 to indicate result validity and 1 to indicate remodeling.

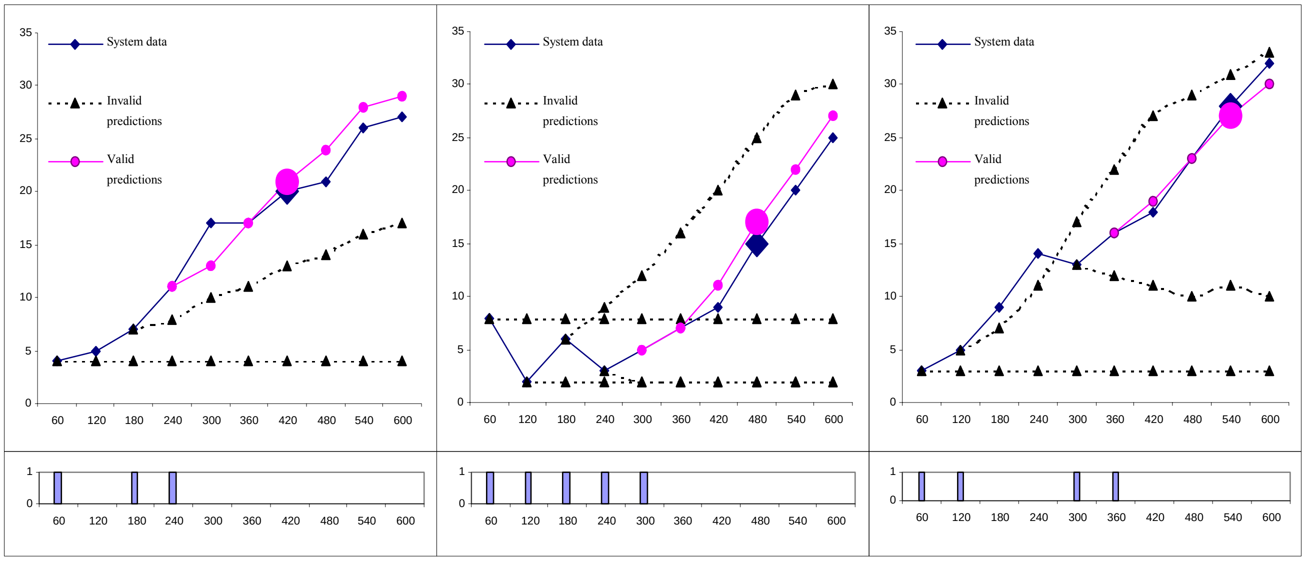

Figure 8: Variation of the number of search requests in the system and model in consecutive intervals

In the first case, invalid predictions are produced at points 60.0 and 180.0, as illustrated in figure 8 (a) by dotted lines. The first series of predictions is determined to be invalid at 180.0; the second at 240.0. Remodeling occurs at 60.0, 180.0 and 240.0. Predictions produced at 240.0 are ultimately considered as reliable at 420.0, after 4 auditing intervals, as previously described. Taking advantage of simulation results, i.e. initiating an additional WebPac server, is thus accomplished at 420.0, before a relatively high number of concurrent requests (i.e. 27, at point 600.0) are issued in the forthcoming intervals. An analogous case appears in figure 8 (b), where simulation results are considered to be reliable at an earlier stage (when there are only 15 search requests), as the number of requests increases more smoothly than in case (a). Remodeling is constantly performed up to point 300.0 and, thus, all previous predictions are discarded. On the other hand, in the third case, a transitory peak occurs at 240.0, but these predictions are later considered as unreliable. The transient behavior of the system causes a delay in taking advantage of simulation results, as these are ultimately considered as valid at 540.0, where the number of search requests is already equal to 26. This is a side effect of using a rather long auditing interval, which makes more difficult to handle frequent system changes. However, using a short interval does not contribute to the validity of statistical observations from the system, as local peaks may be misinterpreted. An analogous misinterpretation may also occur if we decide to reduce the required number of consecutive successful invocations of auditing before predictions are considered to be reliable, which is currently set to 4.

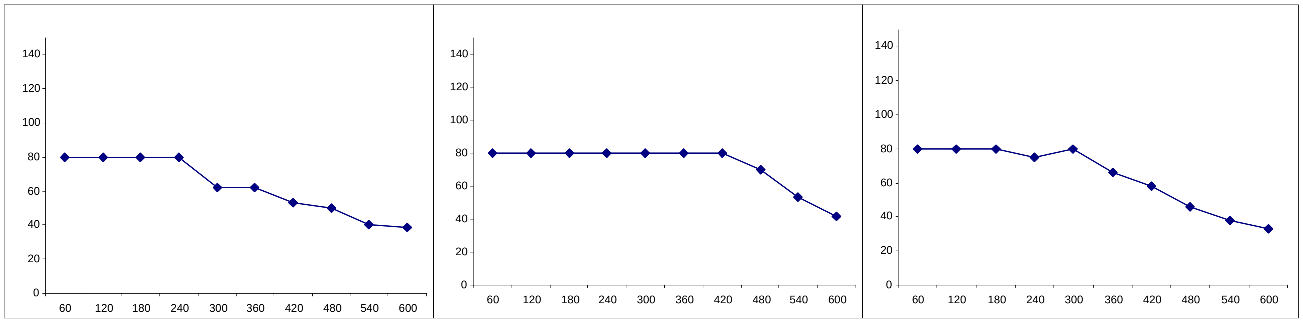
(a) (b) (c)

Figure 9: Variation of the number of replications performed in consecutive intervals

The experiment scheduling method performed well, as replications were executed as expected. The varying number of replications executed within the *AudInt* time frame for the previous three cases is given in figure 9. A maximum of 80 replications has been set. When the number of search requests is increased, the time consumed per replication also increases and thus the number of potential replications is reduced. Due to this, when a small number of requests (<10) is serviced, the number of potential replications may exceed 80. However, when dealing with heavily peaking requests, only a limited number of additional replications may be executed (i.e. $n\text{-}n_o \rightarrow 0$).

6. Conclusions

In FRTS, reaching conclusions for a specific number of intervals ahead imposes that multiple replications have to be made. Timing issues for performing a large number of replications within the given time frame were examined. The method introduced facilitates designing and carrying out experiments consisting of *n* independent replications, where *n* is dynamically determined at runtime, as the duration of replications cannot be efficiently calculated at the design phase. Experimental results from a case study were used to validate the proposed method.

Acknowledgements:

The authors thank the anonymous referees for their comments on this paper.

References

1. Ghosh K., K. Panesar, R. M. Fujimoto, and K. Schwan. "PORTS: A Parallel, Optimistic, Real-Time Simulator," in *Proceedings of 1994 Workshop on Parallel and Distributed Simulation,* IEEE Computer Press, 1994, pp.24-31
2. Cubert R., P. Fishwick, “OOPM: An Object-Oriented Multimodeling and Simulation Application Framework, *Simulation*, vol. 70, no. 6, 1998, pp. 379-395
3. Burns A., A. Wellings, "Hrt-Hood: A structured design method for hard real time systems", *Real Time Systems*, vol. 6, 1994, pp. 73-114
4. Anagnostopoulos D., M. Nikolaidou, "An Object-Oriented Modeling Approach for Dynamic Computer Network Simulation", *International Journal of Modeling and Simulation*, vol. 21, no. 4, 2001
5. Anagnostopoulos D., M. Nikolaidou, P. Georgiadis, "A Conceptual Methodology for Conducting Faster-Than-Real-Time Experiments", *SCS Transactions on Computer Simulation*, vol. 16, no. 2, 1999, pp. 70-77
6. Barros F. J., “Modeling Formalisms for Dynamic Structure Systems”, *ACM Transactions on Modeling and Computer Simulation - TOMACS*, vol. 7, no. 4, 1997, pp. 501-515
7. Zeigler B. P., H. Praehofer, T. Kim, *Theory of Modeling and Simulation (second edition)*, Academic Press, 2000
8. Anagnostopoulos D., “Experiment Scheduling in Faster-than-Real-Time Simulation”, in *Proceedings of ACM/IEEE Workshop in Parallel and Distributed Simulation (PADS 2002)*, IEEE Computer Press, Washington, April 2002
9. Goldsmith S., *A Practical Guide To Real-Time Systems Development*, Prentice Hall, 1993
10. Fujimoto R., “Parallel Discrete-Event Simulation”, *Communications of the ACM*, vol. 33, no. 10, 1990, pp. 30-52
11. Law A.M., W.D. Kelton, *Simulation Modeling and Analysis (third edition),* McGraw-Hill, 2000

12. Ameritech Library Services, *Horizon ver. 5.3. Product Overview*, 1998

13. ISO 23950, *Information and documentation -- Information retrieval (Z39.50) -- Application service definition and protocol specification*, TC 46/SC 4, ISO, 1998