.

INTRODUCING A UML MODEL FOR FASTER-THAN-REAL-TIME SIMULATION

Dimosthenis Anagnostopoulos
George-Dimitrios Kapos

Harokopio University of Athens
70 El. Venizelou Str, 17671
Athens, Greece
email: {dimosthe, gdkapos}@hua.gr

Vassilis Dalakas*
Mara Nikolaidou*

*University of Athens
Panepistimiopolis, 15771
Athens, Greece
email: {vdalakas, mara}@di.uoa.gr

ABSTRACT

Faster-than-real-time simulation (FRTS) is used when attempting to reach conclusions for the near future. FRTS experimentation proves to be the most demanding phase for conducting FRTS, since it requires concurrent monitoring and management of both the real system and the simulation experiments. Having previously introduced a conceptual methodology and specification for conducting FRTS experiments, we now propose an implementation framework, based on the Real Time Unified Modeling Language (RT-UML). The derived RT-UML model includes specific timing attributes and is independent of the application examined via FRTS. Thus, implementation of FRTS program modules can be analyzed and realized, following the guidelines of this model, ensuring the reliability of the results within predetermined time frames. A pilot application regarding FRTS implementation based on the proposed RT-UML model and related experience is also discussed in the paper.

1 INTRODUCTION

Faster-than-real-time simulation is used when attempting to reach conclusions for the near future [1]. 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 a human operator or other agents [2]. Current FRTS research directions involve the distribution of the experiment over a network of workstations, intelligent control [3] and fault diagnosis [4], interactive dynamic simulation [5] and modeling formalisms [6].

In [7] a conceptual methodology for FRTS was described, aiming at providing a framework for conducting experiments dealing with the complexity and the hard real-time requirements. The following simulation phases have been identified: *modeling, experimentation and remodeling*. During experimentation, both the system and the model evolve concurrently and are put under monitoring. Data depicting their consequent states are obtained and stored after predetermined, real-time intervals of equal length, called *auditing intervals*. In the case where the model state deviates from the corresponding system state, remodeling is invoked. This may occur due to system modifications, involve its input data, operation parameters and structure [7]. Modeling issues and formalisms for structure modifications have been thoroughly studied either at the methodological level [8], [9], or for domain/oriented approaches, such as computer networks [10]. To deal with system modifications, remodeling adapts the model to the current system state. This should be accomplished without terminating the real time experiment, that is, without performing recompilation. When model modifications are completed, experimentation resumes. Remodeling can also be invoked when deviations (expressed through appropriate statistical measures) are indicated between the system and the model due to the stochastic nature of simulation, even when system parameters/components have not been modified. Finally, 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 [7].

Experimentation phase comprises monitoring, that is, obtaining and storing system and model data during the auditing interval, and auditing, that is, examining a) if the system has been modified during the last auditing interval (system reformations), b) if the model no longer provides a valid representation of the system (deviations) and, c) if predictions should be used in plan scheduling. Evidently, if conditions (a) or (b) are fulfilled, remodeling is invoked without examining condition (c).

As the system dynamic behavior may result in critical modifications of the system input data, operation parameters and structure, we distinguish three system *reformation* types. Specific measures are monitored to determine whether reformations have occurred. The variables used to obtain the corresponding values are referred as *monitoring variables.* Note that monitoring variables do not follow the single-valued definition of program variables. Auditing examines monitoring variables corresponding to the same real time points (i.e. the current system state and simula-

tion predictions for this point) and concludes for the validity of the model.

Both system and model evolution in real time is depicted in Figure 1. 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 the model predicts the system state at t_n (simulation time equal to t_n) at real time point t_x, we use the notation $Sim(t_x)= t_n$. Auditing is performed at t_{n-1}, t_n, t_{n+1} and, thus, compares states S_x and R_n at time point t_n. If model validity is consecutively ensured within a number of consecutive auditing intervals $[t_{n-2}, t_{n-1}]$, $[t_{n-1}, t_n]$, …, it is likely that simulation predictions are also valid. Thus, plan scheduling is invoked to take advantage of predictions and experimentation resumes.

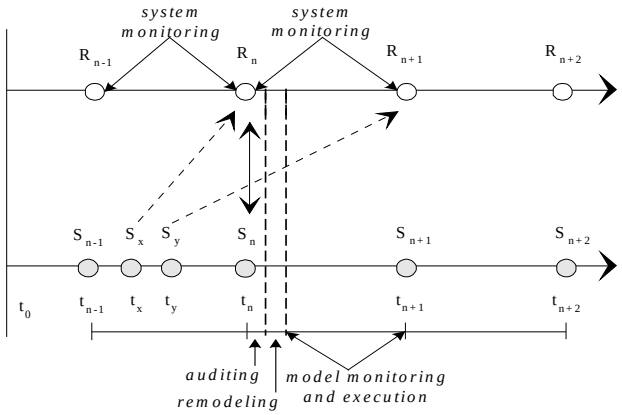

Figure1: Experimentation in FRTS

Experimentation is the most demanding phase of FRTS, since strict time restrictions are imposed: Within an *auditing interval*, model initialization and execution must take place faster than the real system, while auditing and remodeling must be completed within a small fraction of the auditing interval. In fact, experimentation phase can be viewed as a "real time system" itself. In order for an FRTS experiment to be successful, time restrictions should be studied prior to FRTS implementation. Thus, it is essential to provide a model for experimentation activities and their interrelations, facilitating the FRTS researcher to determine the conditions under which such an experiment is feasible, e.g. to determine the auditing interval, the infrastructure need to execute simulation model, e.t.c.

Furthermore, while modeling/remodeling and model execution strongly depend upon the real system, auditing and monitoring are real time activities, which can be implemented based on the same principles for all FRTS experiments [10]. In [11] and [12], a specification of data exchange among simulation components was provided and emphasis was given in activity control and experimental state transition. As simulation activities and control data flows may be the same in diverse FRTS implementations, a common basis for FRTS system development was introduced.

In the following, we introduce a model for FRTS experimentation phase, emphasizing monitoring and auditing activities, which are not domain-oriented. The proposed model aims at establishing common guidelines for FRT simulator development and facilitating its implementation in different platforms according to each researcher's specific need. We decided to adopt UML for FRTS modeling, since it is widely used industry standard and facilitates the automated model implementation in different platforms using a variety of existing tools.Descriptive capabilities of distinct types of UML diagrams are utilized to specify different aspects of FRTS systems: distinct entities and their roles, overall down to detailed logic of FRTS system, synchronized communication, and data specification.

Furthermore, in the proposed specification we use elements from the OMG *UML Profile for Schedulability, Performance and Time Specification* [13] (abbreviated by *Real-Time UML* or *RT-UML*). The profile, also used in [14], enables the detailed specification of critical time and synchronization requirements for FRTS components and an overall performance evaluation. Therefore, we provide a detailed and integrated specification for FRTS systems, leading to standardized implementations of such systems that meet strict time requirements. Implementation may also be facilitated with the use of tools that support code generation given a UML model. This suggests automated program generation and execution during FRTS.

In section 2 we review UML and RT-UML used in the specification of FRTS systems. An overview of the model, emphasizing on the identification of the discrete roles for actors and entities within FRTS, is presented in section 3. FRTS system modeling, focusing on timing issues, is given in section 4. Detailed RT-UML diagrams of FRTS system components specify how each component implements its functionality in terms of events, activities, and actions, emphasizing on timing constraints. In section 5 an simple implementation example is used to illustrate the benefits of RT-UML modeling of FRTS. Finally, in section 6, conclusions are drawn.

2 RT-UML MODELING FRAMEWORK

Unified Modeling Language (UML) [15, 16] is the result of an effort to unify concepts among distinct methodologies, made by the authors of three leading methodologies – Rumbaugh, Booch, and Jacobson. Currently, UML has been adopted as a standard by the Object Management Group (OMG) and is considered a fundamental skill for software engineers.

UML does not provide the required degree of precision (regarding timing issues) for the specification of FRTS. Thus, we use RT-UML [13], which enhances UML diagrams. RT-UML does not propose new model analysis techniques, but it rather enables the annotation of UML models with properties that are related to modeling of time

and time-related aspects. Therefore timing and synchronization aspects of FRTS components are defined and explained in terms of standard modeling elements. RT-UML has a modular structure that allows users to use only the elements that they need. It is divided into two main parts (General Resource Modeling Framework and Analysis Models) and is further partitioned in six subprofiles, dedicated to specific aspects and model analysis techniques. Since the emphasis of this work is on time and concurrency aspects of FRTS systems, we only use elements from the General Time Modeling and General Concurrency Modeling subprofiles.

Each subprofile provides several stereotypes with tags that may be applied to UML models. A stereotype can be viewed as the way to extend the semantics of existing UML concepts (activity, method, class, etc.). For example, a stereotype can be applied on an activity, in order to extend its semantics to include the duration of its execution. This is achieved via a new tag added to the activity, specifying the execution duration. Stereotypes define such tags and their domains.

The proposed FRTS model consists of RT-UML enhanced diagrams, which are annotated according to the conventions used in the RT-UML profile specification and its examples [13]. Stereotypes applied to classes in class diagrams are displayed in the class box, above the name of the class (a in Figure 2). However, when tag values need to be specified for a certain stereotype, a *note* is also attached (b in Figure 2). In sequence diagrams, event stereotypes are displayed over the events, while method invocation and execution stereotypes are displayed in *notes* (c in Figure 2). In activity diagrams, *notes* are also used to indicate the application of a stereotype on an activity, state or transition (d in Figure 2).

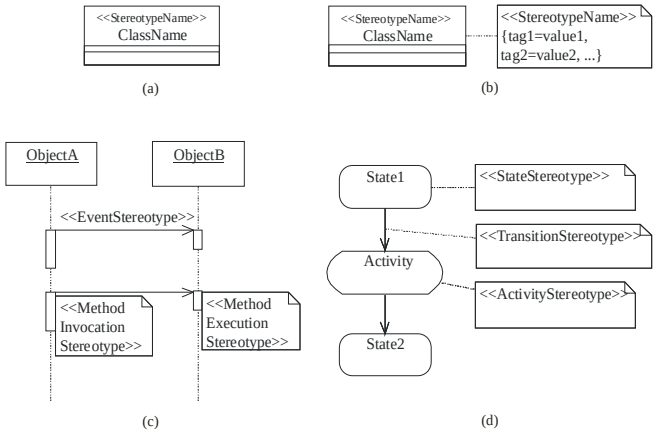

Figure 2: RT-UML notation

The RT-UML stereotypes used in this paper focus on timing, concurrency and synchronization issues, providing considerable precision in the specified model. In class diagrams of this paper we use the CRconcurrent and RTtimer stereotypes. CRconcurrent is used for classes of objects that may be executed concurrently. A CRmain tag holds a reference to the method that should be invoked once the object moves to "executing" state. RTtimer models a timer mechanism. Tag RTduration specifies the duration of the timer mechanism, while RTperiodic indicates whether the timer is periodic or not.

In sequence diagrams we use the RTevent, CRimmediate, CRsynch, CRasynch, RTnewTimer, RTstart and RTaction. RTevent models events of message dispatches, specifying the time instance they occur (through the RTat tag). CRimmediate is also used for message dispatches to indicate that no time is consumed until the message reaches its destination. The CRthreading tag of this stereotype defines the thread that will execute a method (as a result of the message): the thread of the receiver (value "local") or the thread of the sender (value "remote"). CRsynch and CRasynch are used to indicate whether a method is invoked synchronously or not. Stereotype RTnewTimer models methods that create new timers and RTstart is used for events that start timing mechanisms. Finally, RTaction is used for methods, specifying the instance they start (tag RTstart) and their duration (tag RTduration).

In activity diagrams we use the RTaction and RTdelay stereotypes. RTaction was described earlier, while RTdelay is used for pure delay states, specifying their start, end and duration. Table 1 summarizes the RT-UML stereotypes used in the proposed FRTS model, their tags, the concepts applying to, and the diagram types they are used in.

Stereotype	**Tags**	**Applied to**	**Diagram type used in**
RTaction	RTstart, RTend, RTduration	Activity, Method	Activity and Sequence
RTdelay	RTstart, RTend, RTduration	State	Activity
RTevent	RTat	Event	Sequence
RTnewTimer	RTtimerPar	Method	Sequence
RTstart	-	Event	Sequence
RTtimer	RTduration, RTperiodic	Class	Class
CRasynch	-	Method invocation	Sequence
CRconcurrent	CRmain	Class	Class
CRimmediate	CRthreading	Event	Sequence diagram
CRsynch	-	Method invocation	Sequence diagram

Table 1: RT-UML Notation

3 FRTS: A HIGH-LEVEL DESCRIPTION

An object-oriented specification of FRTS is provide in this section. In Figure 3, a UML *use case diagram* is depicted, including all entities involved in FRTS. Both the system and the model, are separate from the main module of FRTS and handled independently. *System environment* (SE) represents the actual system and a surrounding mechanism

facilitating system monitoring. It is considered as a separate entity that interacts with the *FRTS system*. *Model environment* (ME) includes the model and its execution environment (MEE), while the *FRTS System* process is the software module responsible for controlling FRTS. Finally, the user is the actor that enables the whole process, providing the case study.

The *user* provides the *experiment specifications* and manages the *FRTS System* process by starting or stopping the experiment.

System and *model environment* entities provide *raw system data* and *raw model data*, respectivelyThe *FRTS System* process performs auditing to identify potential deviations between the model and the system. In case such a deviation is indicated exceeding a respective remodeling threshold, remodeling is invoked (Remodeling), which results in the construction of a new model that replaces the one currently used (Model management).

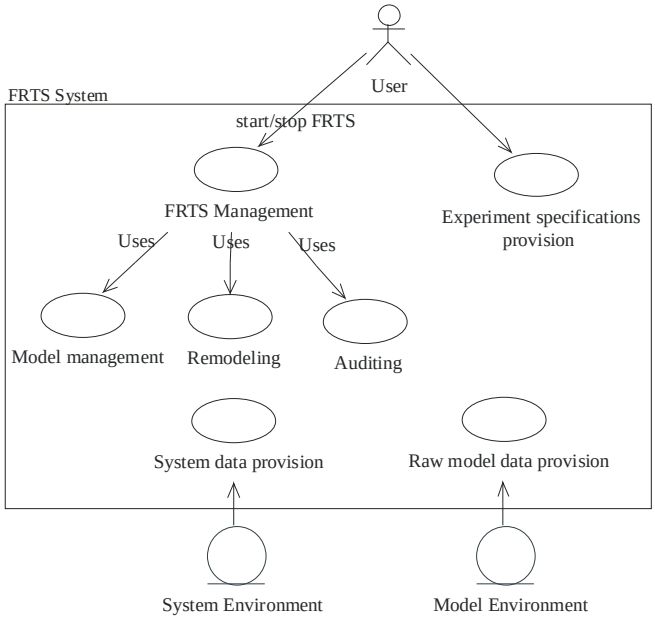

Figure 3: FRTS detailed use case diagram

We focus on the *FRTS System*, as the FRTS coordinating entity. The activity diagram depicted in Figure 4 provides a description of *FRTS System* process. The user is obliged to provide experiment specifications to the process with the *SetExperimentSpecifications* command. Then, s*tart* initiates the experiment, transiting to the *Operational* state.

As previously stated, system monitoring is considered to be performed autonomously by the real system with the aid of expert sensors that store monitoring information. The contribution of *Start System Monitoring* activity is restricted in stimulating the aforementioned sensors to start collecting and recording data by sending the appropriate event to *SE*.

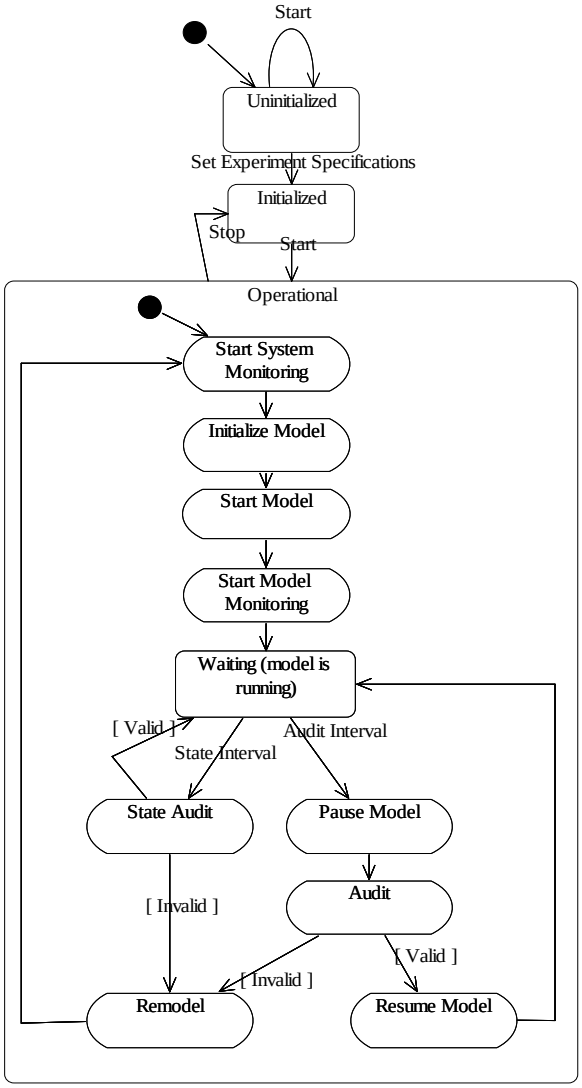

Figure 4: FRTS System activity diagram

Based on the experiment specifications, an initial model is being created (*Initialize Model* activity) using classes from predetermined libraries. Model environment is considered separate from the FRTS environment (e.g. it could be a DEVS-based execution environment). Therefore, *Start Model* activity simply tells *ME* to start simulation and is used for synchronization purposes. Model monitoring is considered to be performed by the *ME* which stores monitoring data. Thus, model and system monitoring are performed concurrently and autonomously, collecting data from both. Model monitoring is executed for a time period equal to auditing interval, such as $[t_{n-1}, t_n]$ in Figure 1, during which the *FRTS System* process mainly remains in state *Waiting* (Figure 4). Model execution is then paused and *Audit* is invoked. *Audit* determines if the model still

provides a valid representation of the system. If invalid, *Remodel* is invoked. Otherwise, *MEE* is informed to resume execution and monitoring of the model.

UML semantics were adequate to represent FRTS system operation in a high-level of detail, since there was no need to represent timing constraints between FRTS specific activities and system/model environment.

In a smaller time interval (*state interval*) than the auditing interval, the *FRTS System* process leaves *Waiting* state, to perform the *State Audit* activity. *State Audit* handles critical, such as structural, modifications of the real system, where remodeling must be performed instantly to restore consistency between the model and the system. Model monitoring is disabled during *Audit* and *Remodel*. On the other hand, system monitoring is never terminated, so that system changes can always be perceived. The only modification it experiences is that it is restarted for synchronization purposes after *Remodel*.

4 FRTS SYSTEM SPECIFICATION

In this section a specification for FRTS systems implementation is provided. First, *FRTS system* main classes and interfaces are presented in a class diagram. Then, FRTS main operations are presented using activity and sequence diagrams. RT-UML semantics are included in the diagrams in both case mainly to indicate concurrent execution of activities, the need for synchronization and timing constraints.

4.1 FRTS Components

Figure 5 depicts the FRTS system design, based on a set of classes and interfaces. The classes are shortly described below (detailed descriptions are given in following subsections):

- *Context* is a utility class, used for storing the experiment specifications, references to the system monitor and the model environment, and monitoring variable values used for state auditing.
- *Control* class initiates the FRTS process.
- *StateAuditor*, *Auditor*, and *Remodeller* are responsible for performing the homonymous operations.
- *Timer* is responsible for producing *StateAudit* and *Audit* events, necessary for triggering *StateAuditor* and *Auditor*.
- Class *UserInterface* is simply the means for introducing user requests and data and therefore, is not further explained.

The following interfaces are also used:

- *IAuditor* interface defines the abstract behavior of an auditor and is implemented via *StateAuditor* and *Auditor* classes.
- *Monitor* interface models the abstract concept of a monitor for variables' values. Interfaces *SystemMonitor* and *ModelExecutionEnvironment* extend this interface to capture specific behavioral characteristics, required for system and model monitoring, respectively.

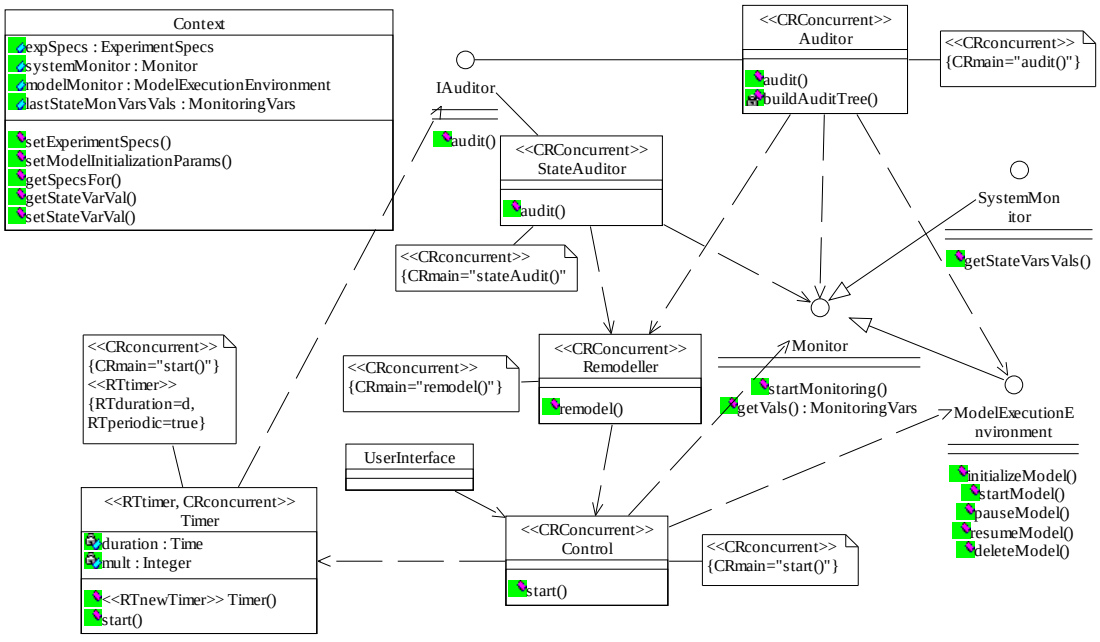

Figure 5: The main FRTS system classes

.

Classes *Control*, *Timer*, *StateAuditor*, *Auditor*, and *Remodeler* are intended to run on separate threads and therefore have the *CRconcurrent* stereotype. Objects of each of these classes operate independently and occasionally concurrently. The *CRmain* tag of CRconcurrent stereotypes indicates the method that is executed when objects of each class are activated. Class *Timer* has also the *RTtimer* stereotype, indicating that it is a timing mechanism that generates an event. Tags *RTduration* and *RTperiodic* further define the behavior of this timing mechanism, specifying its duration and indicating whether it is periodic or not.

No classes are specified for the system monitor and the model environment, since they are not part of the FRTS system. FRTS components require only communication interfaces with the system monitor and the model environment, denoted by *SystemMonitor* and *ModelExecutionEnvironment*.

4.2 Initiation of the FRTS process

Figure 6 shows the sequence of messages exchanged by the FRTS system objects during initiation. This sequence diagram of the FRTS process starts when the user sends the *start()* event to the *Control* (through the *UserInterface*) at a random time instance t_y. The *start()* event causes the immediate execution of the homonymous method of the *Control*, as indicated by the *CRimmediateExecution* stereotype. Value *'local'* of the tag *CRthreading* shows that the *start()* method of *Control* is not executed by the thread of the invoking object (*UserInterface*), but by a separate, local thread of the *Control*. A *'remote'* value on this tag would indicate execution of the method by the thread of the invoking object. The *CRasynch* stereotype indicates that the invocation of the *start()* method is asynchronous, i.e. the invoking object does not wait for the execution of the method to be completed. At this stage several initiation messages are exchanged until the FRTS process reaches its stable state of periodic audits and state audits. This happens when the last message (*start()*) is sent to the *Timer* that will repeatedly produce state audit and audit events from this point on. All method executions are annotated with the appropriate *RTaction* stereotypes that indicate when each execution starts (tag *RTstart*) and its duration (tag *RTduration*).

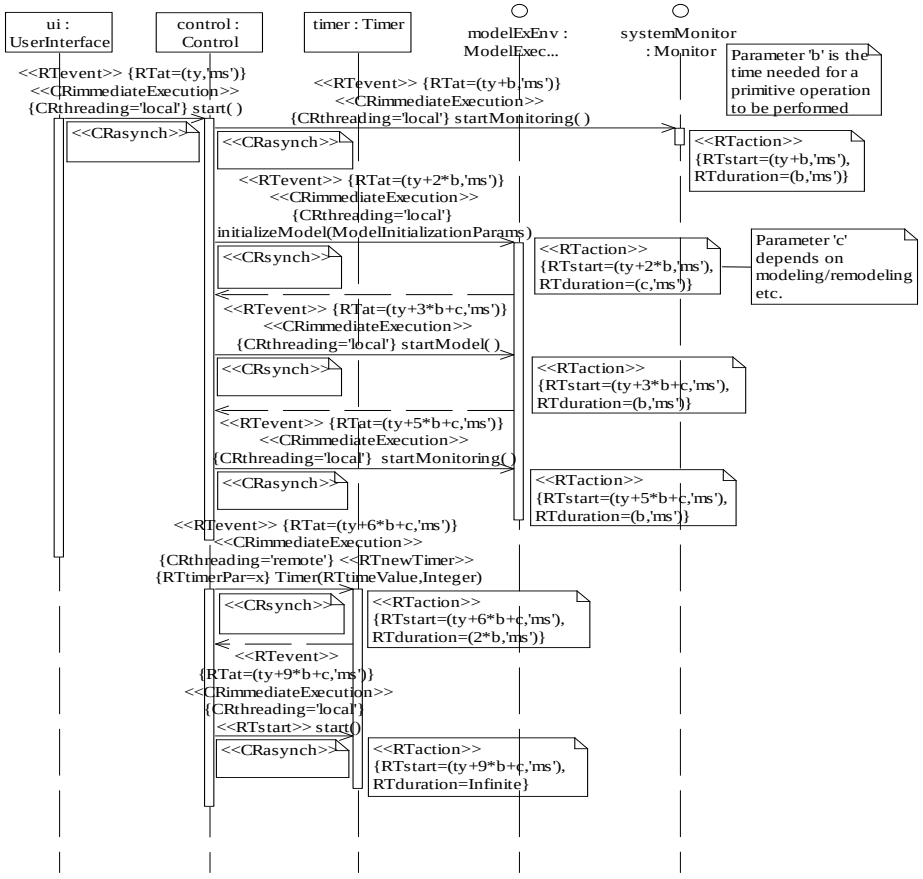

Figure 6: Sequence diagram for starting the FRTS process

The use of RT-UML in sequence diagram of Figure 6 clarifies thread synchronization and execution, determines event occurrence and action duration, and enhances its semantics. Thus, an in-depth and comprehensive view of the FRTS system is obtained.

The activity diagram of Figure 7 defines the functionality of the *start()* method of class Control. Each activity of the diagram is annotated with the appropriate RTaction stereotype note. Using this kind of stereotype and its RTduration tag, activities' durations are specified. The lower part *(do/)* of each activity defines the actions executed or messages sent. Message dispatches are denoted with the ^ symbol. The overall duration of *start()*method is *9*b+c ms*, where b is the time needed for a basic operation to be performed (arithmetic operation, method invocation, etc.). Parameter c is the duration of model's initialization and depends on the experiment specification. The overall duration of *start()* refers to the duration from the time instance when the user sends a *start()* event until everything has been initialized and *Timer* is started.

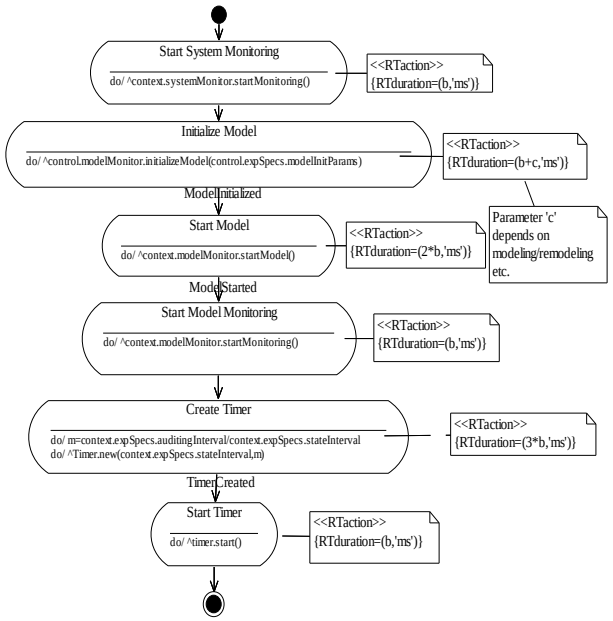

Figure 7: Activity diagram for method *start* of class *Control*

4.3 Audit

Audit is the key experimentation activity determining model validity through comparing the corresponding system and model monitoring variables. Auditing is activated either after a *state interval* or an *audit interval*. Two distinct cases are thus considered: standard auditing and state auditing. Throughout this paper, the term *auditing* refers to standard auditing. *State auditing* is explicitly referenced.

During auditing, system modifications, involving its input data, operation parameters and structure, as well as deviations between the system and the model are examined to determine model validity. If remodeling is required, a *remodeling indication* is produced. All monitoring variables are used in this process.

Monitoring variable comparison is realized using the auditing tree, which is a conceptual tree structure. It is divided into two subtrees and includes two corresponding types of end nodes, *OR* and *AND*, as depicted in Figure 8. The audit activity constructs the auditing tree retrieving system and model monitoring variable entries from the *System Monitor* and *Model Execution Environment*, respectively.

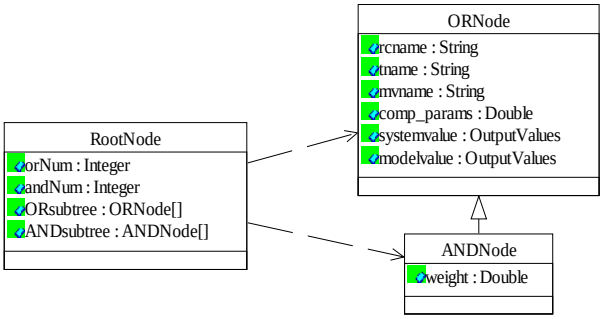

Figure 8: Auditing tree class diagram

Both Audit and State Audit execution are restricted by strict timing concerns, since in both cases the auditing tree must be constructed in a small fraction of the audit/state audit interval. Furthermore, the auditing tree construction is bounded by system and model environments since monitoring variable values must be fetched from both of them. These restrictions are denoted in detail in corresponding sequence and activity diagrams, where RT-UML use offers the ability to estimate the time elapsed in separate activities or the whole auditing process in total. Hence, bottlenecks regarding the execution time of specific Auditing and Model/System Environment processes (e.g. comparing values of a monitoring variable) may be identified during analysis and Auditing implementation performance can be measured and validated with regard to Model/System Environment operation. For example, since the FRTS Modeler is able to realize the way the overall duration of audit depends on the number of monitoring variables or the fetching mechanism of System Environment, he/she may regulate the operation of all FRTS modules.

In figures 9 and 10, the State Audit RT-UML sequence and activity diagrams are presented. As shown in figure 9, *state audit* activity inspects the current system state to determine if reformations have occurred. In this case, the model no longer provides a valid representation and the relevant *remodelling indication* is produced. As indicated in the activity diagram in figure 10, only variables designated as *state monitoring variables* are retrieved dur-

ing *state audit*. Each of these variables is compared to its previous known value and the newer is stored. If the deviation between the two values supersedes the specified *compParam*, it is considered as invalid and the algorithm directly invokes remodeling to modify the model with minimum time overhead, without exhaustively examining the remaining state monitoring variables. Otherwise, the state auditor examines the remaining state monitoring variables.

As indicated in Figure 10, the overall duration of the *state audit* is *8*b+net1+f ms*, where *f* belongs in *[4*b+d, (4*b+d)*e]*, *d* is the mean time for the comparison for one variable and *e* is the number of state monitoring variables.

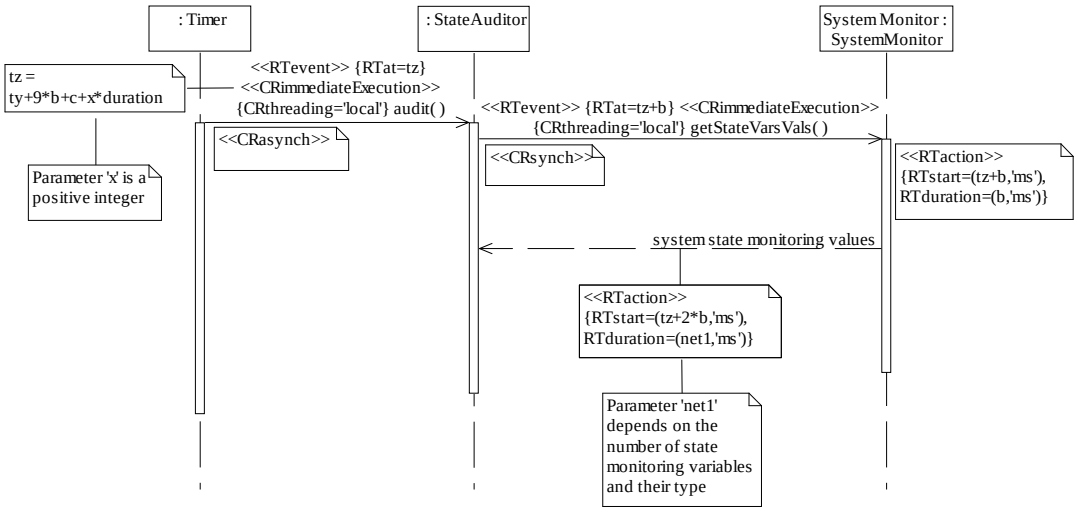

Figure 9: State audit sequence diagram

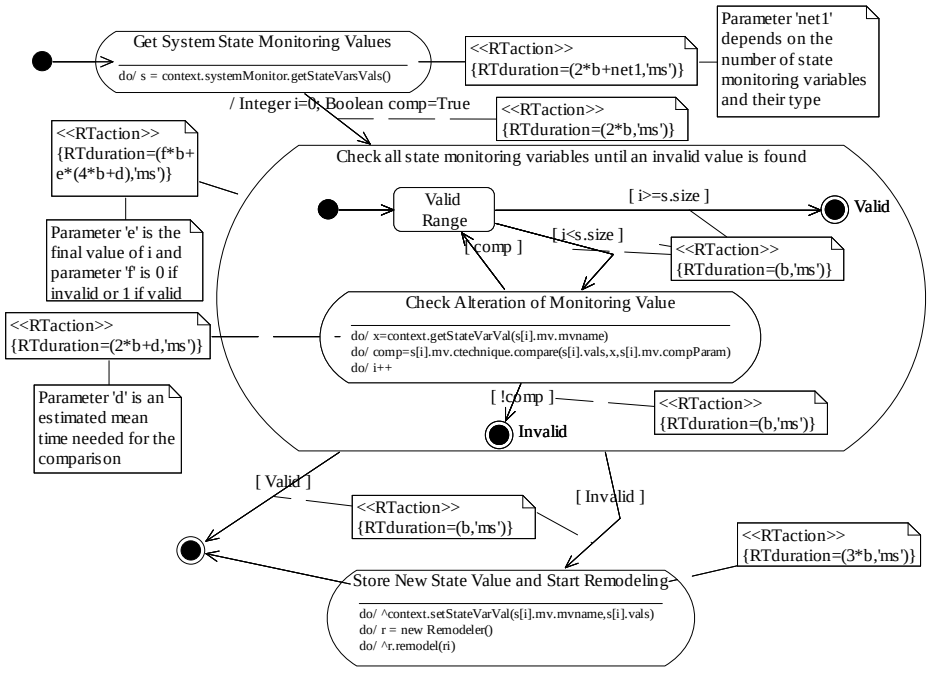

Figure 10: Activity diagram for method *audit* of class *StateAuditor*

.

5 IMPLEMENTING A SIMPLE FRTS USING THE RT-UML MODEL

In order to evaluate FRTS RT-UML model richness and flexibility, a simulator was built in Java using the diagrams analytically discussed in sections 3 and 4. To contact FRTS experiments successfully, the execution time of specific activities must be similar to the time estimations reported in the model. A simple FRTS experiment system is presented in the following for evaluation purposes. The Simulation Environment (SE) and simulation models were constructed in Java, as well. The simulator was easily implemented using Rational Rose platform (programming effort was minimized).

In the following, FRTS is applied in a two node web site, where the second node is used only in cases of heavy load (that is when FRTS predicts that each node load is over a certain threshold). Suppose that visitor enquiries are two kind of processing jobs J_1 and J_2 that fill two separate queues Q_1 and Q_2 respectively. Each job has an inter-arrival time λ_i and a predetermined service time ε_i ($\varepsilon_1 \geq \varepsilon_2$). Both queues are connected with a server S_i as illustrates Figure11. Thus, the web site can be modeled as a Multi-Queue, Multi-Server System.

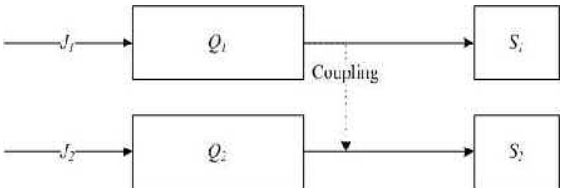

Figure 11: Example topology

Denoting as d_i, the average queue delay we may define the following scenario for our case study. In the beginning each server serves only its associated queue (Coupling does not exist). However, if $d_1 \geq M_1$ and $d_2 \leq M_2$ we activate the coupling among the first queue Q_1 and the second server S_2, activating a mechanism that enables S_2 to serve one job from queue Q_1 each time its queue (Q_2) is empty. This mechanism is deactivated in case $d_1 \leq M_1$ or $d_2 \geq M_2$. Then again, each server serves only its associated queue (Coupling does not exist).

In order to conduct the experiment, detailed description of Monitoring Variables and Remodeling Conditions was needed during the initialization phase. As model initialization parameters, the following variables were used:

- 2 job types
- 2 queues
- 2 servers
- Inter-arrival parameter for job 1 = 3
- Inter-arrival parameter for job 2 = 3
- Service time for job 1 = 10 sec
- Service time for job 1 = 1 sec
- Average queue delay in queue 1 = 60 sec
- Average queue delay in queue 1 = 5 sec

thus, we have *modelInitParams = (2, 2, 2, 3, 3, 10, 1, 60, 5).*

Having the model variables and the initialization parameters the MEE can now build the model and execute it.

Checking values for both system and model of these monitoring variables during auditing we apply remodeling following the described scenario and in case a server is down.

The scenario just described is only a case study needed to be executed with our FRT Simulator implementation to test the validity of the proposed framework. Table 2 presents the results of the experimentation with the example described in this section and the FRTS simulator we built. Experimentation was conducted within Sun's Netbeans IDE and the Netbeans Profiler plugin. Measured times (third column) are presented against estimated durations (second column) by the RT-UML FRTS model analysis. The table contains the most important time periods:

a) Execution time of a basic operation. It entirely depends on the computer configuration where the experimentation is conducted. No theoretical estimation can be made. The measured value is substituted in the formulas that estimate other time periods.

b) Audit and state audit intervals.

c) Audit and state audit durations. A fundamental requirement is that state audit duration is less than state audit interval.

For each time period both estimated and measured, an average, a minimum and a maximum value are given.

Duration (avg,min,max) in msec	**Theoretical Estimation**	**Measured Time**
Time for basic operation	Computer depended (b)	0.0377,0.0011, 0.8229
Audit interval	5000, 5000, 5000	5004.09, 4871, 5278
Audit duration	5.479, 0.155, 123.441	2.700, 0.090, 53.400
State audit interval	1000, 1000, 1000	999, 891, 1106
State audit duration	0.565, 0.017, 12.344	0.233, 0.087, 10.70

Table 2: Basic FRTS time attribute comparison

As far as the estimated time periods are concerned, audit and state audit intervals are explicitly defined rather than estimated. Also, since the basic operation duration (b) is not estimated, but the measured time is used in other formulas.

State audit duration is estimated by the formula *8*b+net1+f*, where *f* belongs in *[4*b+d, (4*b+d)*e]*, *d* is the mean time for the comparison for one variable and e is the number of state monitoring variables. In our example there is only one state monitoring variable (*e=1*) and the mean time for the comparison of the integer variable is two basic operations (*d=2*b*). Also, as there isn't any factor that would introduce delays in the reception of state monitoring variable values from the system, parameter net1 can

be estimated to be equal to one basic operation duration (*net1=b*). Therefore, the formula estimating the state audit duration becomes *15*b*. The respective cell is filled using the measured value for the basic operation duration (*b*).

Similarly, for the estimation of the audit duration, the formula *14*b+g+net2+net3+k* is used. Parameter g is the time for the audit tree to be built, *net2* and *net3* depend on the number (*h=9*) and the type of the monitoring variables, and *k* belongs in *[b+h*(4*b+d), b+h*(5*b+d)]*. Parameter *g* can be estimated to be *6*b* times the number of monitoring variables (*6*b*h=54*b*). Like *net1* in state audit duration, *net2* and *net3* are considered to be equal to *h*b=9*b* each. Considering that *h=9* and *d=2*b*, *k* belongs in *[55b, 64*b]*.Therefore, audit duration belongs in [*141*b*, *150*b*]. The respective cell is filled using the measured value fot the basic operation duration (*b*).

Comparing the theoretical estimations with the measured times in table 2, the following conclusions are reached: a) audit and state audit intervals are quite accurate, b) estimated audit and state audit durations are comparable to the measured ones. Also, estimations for maximum audit and state audit durations are higher than the measured ones, indicating that the estimated maximum values may be used as the lower limit for audit and state audit duration.

6 CONCLUSIONS

The main objective of the work presented in this paper was to introduce a specification for FRTS experimentation, which was not domain-oriented and establishes common guidelines for developing FRT simulators. We adopted RT-UML to provide a thorough and complete model for FRT simulators emphasizing timing and concurrency issues. RT-UML enabled the description of time constraints imposed in FRTS, while modeling process was straightforward, and no extensions were needed to describe FRTS. Detailed RT-UML diagrams specify how each FRTS component operates in terms of events, activities, and actions and infers estimations about time consistency and overall behavior of specific FRTS simulators. The behavior of FRTS simulators, apart from their implementation, strongly depends on the application domain and the experiment specifications used. Thus, time consistency of FRTS simulators may be completely justified only in the context of an application domain and specific experiment specifications. To this direction the proposed model quantified this interdependence and facilitates the evaluation of FRTS simulators in certain contexts.

7 REFERENCES

[1] Cleveland J. et al., *Real Time Simulation User's Guide: The Red Book*, Analysis and Simulation Branch, NASA Langley Research Center, 1997

[2] Fishwick P., "OOPM/RT: A Multimodelling Methodology for Real-Time Simulation", *ACM Transactions on Modelling and Computer Simulation*, 9(2), 1999

[3] Cai Z., Y. Wang, J. Cai, " A Real-Time Expert Control System", *Artificial Intelligence and Engineering*, Elsevier Science, vol. 10, 1996, pp. 317-332

[4] Norvilas et al., "Intelligent Process Monitoring by Interfacing Knowledge-Based Systems and Multivariate Statistical Modelling", *Journal of Process Control*, Elsevier Science, 2000, pp. 341-350

[5] Tyreus B.D., "Interactive Dynamic Simulation using Extrapolation Methods", *Computer and Chemical Engineering*, vol. 21, 1997, pp. 173-179, Pergamon Press

[6] Zeigler B. P., H. Praehofer, T. Kim, *Theory of Modeling and Simulation (second edition)*, Academic Press, 2000

[7] Anagnostopoulos D., M. Nikolaidou, "Timing Issues and Experiment Scheduling in Faster-than-Real-Time Simulation", *Simulation: Transactions of the Society for Modeling and Simulation International*, Vol. 79, No 11, SCS, 2003

[8] Barros F. J., "Modelling Formalisms for Dynamic Structure Systems", *ACM Transactions on Modelling and Computer Simulation - TOMACS*, 7(4), pp. 501-515, 1997

[9] Zeigler B.P, H. Praehofer, "Systems Theory Challenges in the Simulation of Variable Structure and Intelligent Systems", in *CAST: Computer Aided Systems Theory, Lecture Notes*, Springer-Verlag, Berlin, pp: 41-51, 1997

[10] Anagnostopoulos D., M. Nikolaidou, "An Object-Oriented Modelling Approach For Dynamic Computer Network Simulation", *International Journal of Modeling and Simulation*, Vol. 21, No 4, Acta Press, 2001

[11] Goldsmith S., *A Practical Guide to Real-Time Systems Development*, Prentice Hall, 1993

[12] Anagnostopoulos D., Dalakas V., Nikolaidou M., Vescoukis V., *A Structured-Analysis-Based Specification for Faster-than-Real-Time Simulation*, in Systems Analysis Modeling and Simulation Journal (SAMS), Taylor and Francis

[13] UML Profile for Schedulability, Performance, and Time Specification, version 1.0, available on-line at `http://www.omg.org/docs/formal/03-09-01.pdf`

[14] Bertolino A., Marchetti E., Mirandola R., *Real-Time UML-based Performance Engineering to Aid Manager's Decision in Multi-project Planning*, in Proceedings of WOSP 2002

[15] Rumbaugh J., Jacobson I., Booch G., *The Unified Modeling Language Reference Manual*, Addison Wesley, 1998

[16] OMG Unified Modeling Language Specification, version 1.5, available on-line at `http://www.omg.org/docs/formal/03-03-01.pdf`