.

ESTABLISHING A COMMON MODELING FRAMEWORK USING UML TO EFFECTIVELY SUPPORT FASTER-THAN-REAL-TIME SIMULATION

Mara Nikolaidou*
Vassilis Dalakas*

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

Dimosthenis Anagnostopoulos
George-Dimitrios Kapos

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

Abstract

Faster-than-real-time simulation (FRTS) is used when attempting to reach conclusions for the near future. FRTS experimentation phase deals with the complexity and hard real-time requirements imposed by the concurrent execution of the real world system and corresponding simulation model. It is important for the FRTS experiment designer to study timing constraints, thus to set limitations in the execution time of specific activities. As simulation activities and control data flows may be the same in diverse FRTS implementations, a common basis for FRTS system development is introduced based on the Real Time Unified Modeling Language (RT-UML). The proposed approach enables the detailed specification of critical time and synchronization requirements for FRTS components and an overall performance evaluation. The detailed specification for FRTS systems leads to standardized implementations of such systems that meet strict time requirements.

1. INTRODUCTION

Faster-than-real-time simulation (FRTS) aims at studying the behavior of real-world systems in the near future [1]. In this type of simulation, simulation runs concurrently with the real world system and advancement of simulation time occurs faster than real world time. Simulation model interacts with the real world system during experimentation in order to test model validity and adjust to system changes. Although constructing models for FRTS is a challenging tasks, this issue has already been resolved [2,3]. FRTS experimentation phase deals with the complexity and hard real-time requirements imposed by the concurrent execution of the real world system and corresponding simulation model, thus it should be emphasized. FRTS experimentation phase can be viewed as a real-time system itself, thus is should be modeled and thoroughly studied. Structured analysis [4] and real-time system specification techniques [5], have already been used for this purpose. In both cases, a detail specification of simulation activities was provided, while 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 should be introduced. Since simulation activities, especially during experimentation, must be executed in real time, it is important for the FRTS experiment designer to study timing constraints, thus to set limitations in the execution time of specific activities, and explore concurrency issues, thus model activity synchronization. An estimation of the duration of discrete activities and the dependencies between them may facilitate the FRTS designer to decided whether FRTS experiments are visible and identify constraints imposed be the real time system itself and the simulation model. Simulation model constraints may originate from the model itself (e.g. how fast the model may run) or infrastructure resources (e.g. how fast is the hardware supporting).

In order to explore these issues and achieve a consistent transition from FRTS specification to the implementation of the corresponding program modules, we provide a detailed and multi-facet specification using UML. Elements from the OMG UML Profile for Schedulability, Performance and Time Specification [6] (abbreviated by Real-Time UML or RT-UML) are used to depict activity synchronization and timing constraints. The proposed approach enables the detailed specification of critical time and synchronization requirements for FRTS components and an overall performance evaluation. The detailed specification for FRTS systems leads 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.

A brief introduction to the FRTS methodology emphasizing experimentation phase modeling requirements is presented in section 2. An overview of FRTS model focusing on timing issues, is presented in section 3. Detailed RT-UML diagrams, emphasizing Auditing activity, specify how each component implements its functionality in terms of events, activities, and actions, all of which have precise time orientation. In section 4 an example is used to illuminate the presented model. Conclusions reside in section 5.

2. FRTS METHODOLOGY AND REQUIREMENTS

In [7] a conceptual methodology for FRTS was described, aiming at providing a framework for conducting experiments dealing with complexity and hard real-time requirements. The following simulation phases have been identified: *modeling, experimentation and remodeling*. In order to conduct FRTS experiments, it is assumed that a model of the real-world system in the proper level of detail can be constructed. 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*. Experimentation comprises *monitoring*, that is, obtaining and storing system and the model data during the auditing interval, and *auditing*, that is, comparing them at the end of every auditing interval . During auditing the following conditions are 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). In both cases, *remodeling* is invoked. 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 [10]. Evidently, if conditions (a) or (b) are fulfilled, remodeling is invoked without examining condition (c) (0).

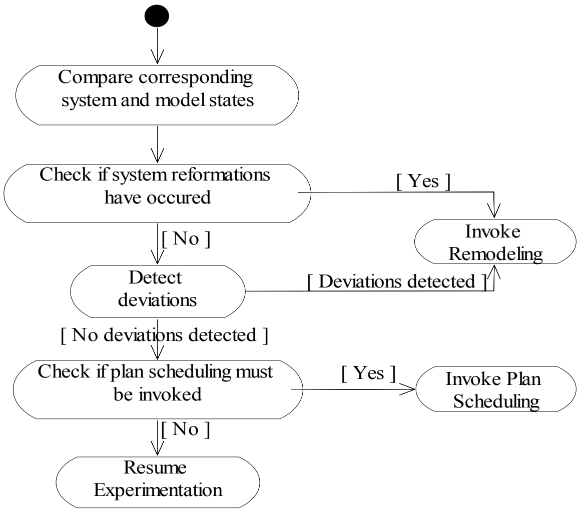
Compare corresponding system and model states
Check if system reformations have occured
[Yes]
Invoke Remodeling
[No]
Detect deviations
[Deviations detected]
[No deviations detected]
Check if plan scheduling must be invoked
[Yes]
Invoke Plan Scheduling
[No]
Resume Experimentation

Figure 1: Auditing control activity diagram

The system and the model are compared based on their corresponding states. *State* is a set of attributes describing the model and the system at specific time instances. Attributes are defined as *Monitoring Variables* and describe the system structure, operation parameters and input data [7]. 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 simulation predictions for this point) and concludes for the validity of the model.

Modeling issues and formalisms for system reformations have been thoroughly studied either at the methodological level [8], [3], or for domain/oriented approaches, such as computer networks [9]. To deal with system reformations or system/model deviations, 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. Both system reformations (e.g. addition of a network node) and system/model deviations (e.g. significant deviation of network throughput) are modeled using Monitoring Variables.

To accomplish FRTS experimentation phase should be emphasized. Both system and model evolution in real time is depicted in 0. 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.

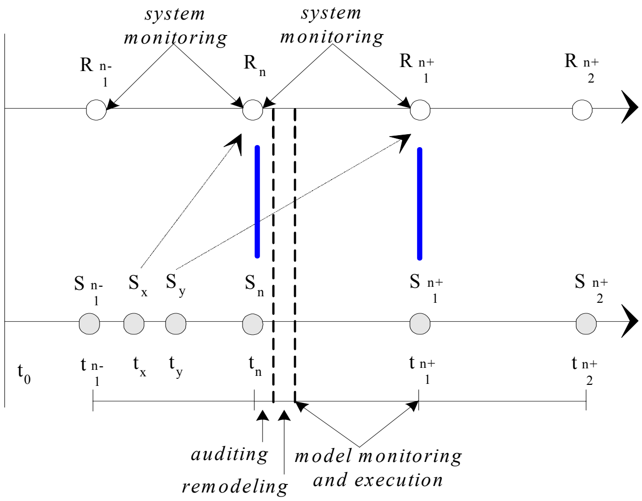

Figure 2: Experimentation in FRTS

As indicated in the figure, the following activities must accomplished during auditing interval: auditing (duration TAudit), remodeling (if needed) (duration TRemodel), model initialization (in case of remodeling) (duration TInit) and model execution (duration TExec). Thus, timing issues should be addressed to effectively support FRTS, since:

`TAudit+TRemodel+TInit+TExec<AudInt` (Auditing interval)

Futhermore, auditing and remodeling must be executed real fast, that is:

`TAudit, TRemodel <<TExec`

All aforementioned activities are interrelated. It is obvious, that in order to conduct FRTS experiments, one should ensure their duration in comparison to the duration of others and the definition of the auditing interval. Thus, it is useful to identify the dependencies and restrictions regarding each activity. For example, model initialization strongly depends on model definition and model execution environment, while auditing duration may relay to the number of Monitoring Variables compared.

Prior implementation, FRTS experiment designer should have the opportunity to study timing constraints and activity duration dependencies, to decide the conditions under which FRTS is feasible and the cost of experimentation. Since FRTS is a "real time" system itself, there is a need to establish a formal framework of the specification of FRTS providing the required degree of precision regarding timing issues. We propose to do so using OMG *UML Profile for Schedulability, Performance and Time Specification* [6] (abbreviated by *Real-Time UML* or *RT-UML*). The profile, also used in [10], enables the detailed specification of critical time and synchronization requirements for FRTS components and an overall performance evaluation. Thus, FRTS user may analyze of the factors affecting the experiment prior implementation.

FRTS experimentation phase is usually supported by custom FRT Simulators built especially for a specific experiment. They usually have object-based architecture and they facilitate the management of FRTS experiment. They must interact with the real-world system, perform monitoring and auditing tasks and may or may not include the simulation model execution environment.

Modeling/Remodeling and Model Execution are system specific, while Monitoring and Auditing are common to all FRT Simulators. The proposed UML framework of FRTS specification results in the creation of an FRT Simulator model, establishing common guidelines for FRT Simulator development. The FRT simulator model is not domain-oriented, emphasizes monitoring and auditing and provides specifications for the interaction with the real-system and the simulation model execution environment.

3. FRTS MODELING

An object-oriented specification of FRTS is provided in this section. We adopted UML for modeling purposes, since it is widely used in software engineering and facilitates FRTS experiment designers to built their own implementation of the model in different platforms using a variety of existing tools. Different diagrams may be used to describe different aspects of Experimentation. Especially *Sequence* and *Activity diagrams* were proven very useful. Sequence diagrams used to study the interaction between Experimentation activities and sub-activities, while Activity diagrams provide detail description of activities/subactivies. Both the system and the model, are separate from the main module of FRTS and can be viewed as autonomous systems. *System environment* (SE) represents the actual system and a surrounding mechanism responsible for monitoring. *Model environment* (ME) includes the model and its execution environment (MEE), while the *FRTS System* process is the software module responsible for controlling FRTS. The sequence diagram in 04 emphasizes the communication between the aforementioned entities (*User*, *FRTS System*, *ME* and *SE*), in terms of message exchange. UML semantics were adequate to represent FRTS in a high-level of detail, since there was no need to represent timing constraints between FRTS specific activities and system/model environment.

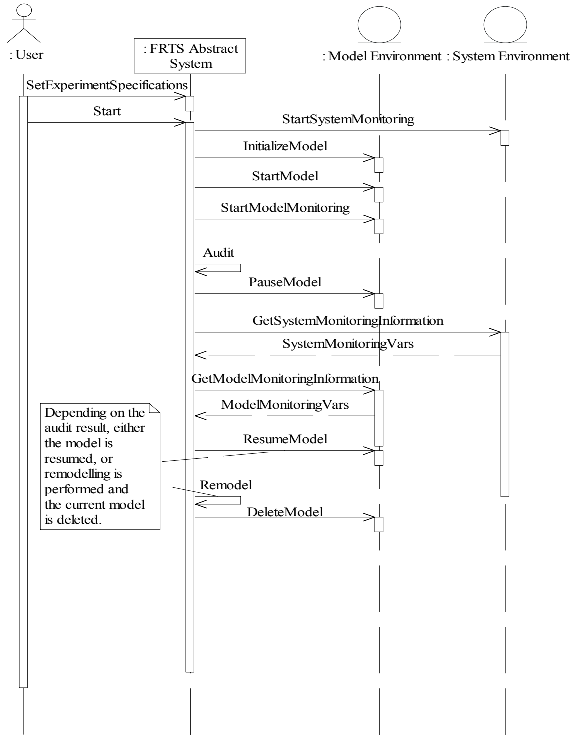

Figure 3: FRST sequence diagram

In a more detailed level, it is important to understand the feasibility of the experiment, thus identify concurrent activities, estimate the duration of each activity and identify synchronization requirements and time constraints. RT-UML [6] 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. 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 sub-profiles. Each sub-profile provides stereotypes with tags that may be applied to UML models. For example, a stereotype with a new tag can be applied on an activity, in order to extend its semantics to include the execution duration. In sequence diagrams, event stereotypes are displayed over the events, while method invocation and execution stereotypes are displayed in *notes*. In activity diagrams, *notes* are also used to indicate the application of a stereotype on an activity, state or transition.

The definition of RT annotated UML diagrams describing FRTS activities identified in figure 4 (message representation) leads to standardized implementations that meet strict time requirements. In the following, we focus on *Auditing* activity to identify the benefits of the proposed approach.

3.1. Auditing Specification

Auditing is the key experimentation activity determining model validity through comparing corresponding system and model monitoring variables. All monitoring variables participate in this process. Monitoring variables are defined as part of the experiment specification. The corresponding class diagram is depicted in figure 4. As shown in the diagram (*Indicator* class), monitoring variables correspond to system structure operation parameters, input data and system/model deviation indicators. Each *MonitoringVariableSpec* object (MV_i) has two corresponding objects: $SystemMV_i$ and $ModelMV_i$ (of class *MonitoringVar*) for the values deriving from the system and the model, respectively. $SystemMV_i$ is calculated as a function of either a single-valued variate (performance measure or system parameter) or multiple system observations $R_{i1}, R_{i2}, \ldots$, and in this case $SystemMV_i = f_i (R_{i1}, R_{i2}, \ldots)$. $ModelMV_i$ can also be calculated as a function of either a single-valued variate or an output stochastic process. As n replications are executed, $ModelMV_i$ is calculated as a function of n stochastic processes [11]. In FRTS, the number of observations per run is not the same, as simulation ends at a specific simulation time point, without considering the current status of system entities. Replication results are thus extracted from $k_1, k_2, \ldots, k_n$ observations. The output process of each replication produces a single statistical sample $S_{ij} = g(S_{ij1}, S_{ij2}, \ldots, S_{ijk_j})$. If the difference a specific monitoring variable values supercedes the specific “compParam”, a *remodeling indication* is produced and remodelling is invoked.

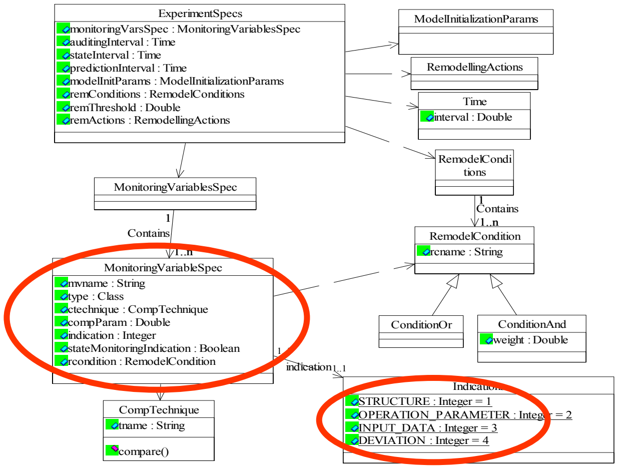

Figure 4: FRTS Data Specification

Auditing must be completed within a small fraction of the auditing interval. RT-UML modeling facilitates:

- Detail estimation of Audit duration
- Identification of factors/ processes affecting it and their relationships
- Comparing it to other crucial time-related attributes, e.g. auditing interval.

The sequence diagram of figure 5 describes the sequence of messages exchanged by the FRTS system objects during audit. *Auditor* (responsible for performing auditing) is invoked by Timer to perform auditing at the end of each auditing interval. To indicate periodical invocation of activities, the *RTevent* RT-UML stereotype is used. RTevent models events of message dispatches, specifying the time instance they occur (through the RTat tag). In this case, at time instance t_w the *Timer* produces an *audit* event and then the *Auditor* pauses the model execution with *pauseModel()* (at t_w+b, where b is the time needed to perform a basic operation) and retrieves monitoring variable values from the system and the model with *getVals()* method invocations (at t_w+3b). Finally, it performs comparisons between system and model values within a time interval dependent on the auditing tree of the experiment. *CRimmediate* stereotype is used 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* note stereotypes are used to indicate whether a method is invoked synchronously or not. In this case, auditor is asynchronously invoked by timer at predefined time intervals, while auditor must wait for the completion of each method to resume its execution. Finally, *RTaction* note stereotype is used for methods, specifying the instance they start (tag RTstart) and the estimation of their duration (tag RTduration). *pauseModel()* action duration is estimated to be b (time needed to perform a basic operation), while the duration of sending the response message back to the auditor is also estimated to be b. *getVals()*action duration is estimated to be b (time needed to perform a basic operation), while the duration of sending the model monitoring variable values to the auditor is estimated as *net3,* depending on the number and structure of monitoring variables.

Figure 6 illustrates auditing activity. Monitoring variable comparison is realized using the auditing tree [11], a conceptual tree structure. Prior constructing the auditing tree, auditing activity must retrieve system and model monitoring variable entries from the *System Monitor* and *Model Execution Environment*, respectively. Each monitoring variable comparison corresponds to a single node of the auditing tree. The type of the each end node is determined by the type of the *RemodelingCondition* object related to the monitoring variable. End nodes of type *OR* represent comparisons that autonomously – if fulfilled – cause remodeling. Nodes of type *AND* are aggregately evaluated to determine if remodeling is required.

Accessing all nodes, we insure that all remodeling conditions are evaluated prior to the initiation of remodeling and all reformations/deviations are detected, so that appropriate remodeling actions can be considered. The auditing algorithm concludes that the model is invalid if at least one of the *OR* node conditions or the aggregate evaluation of the *AND* node conditions are fulfilled.

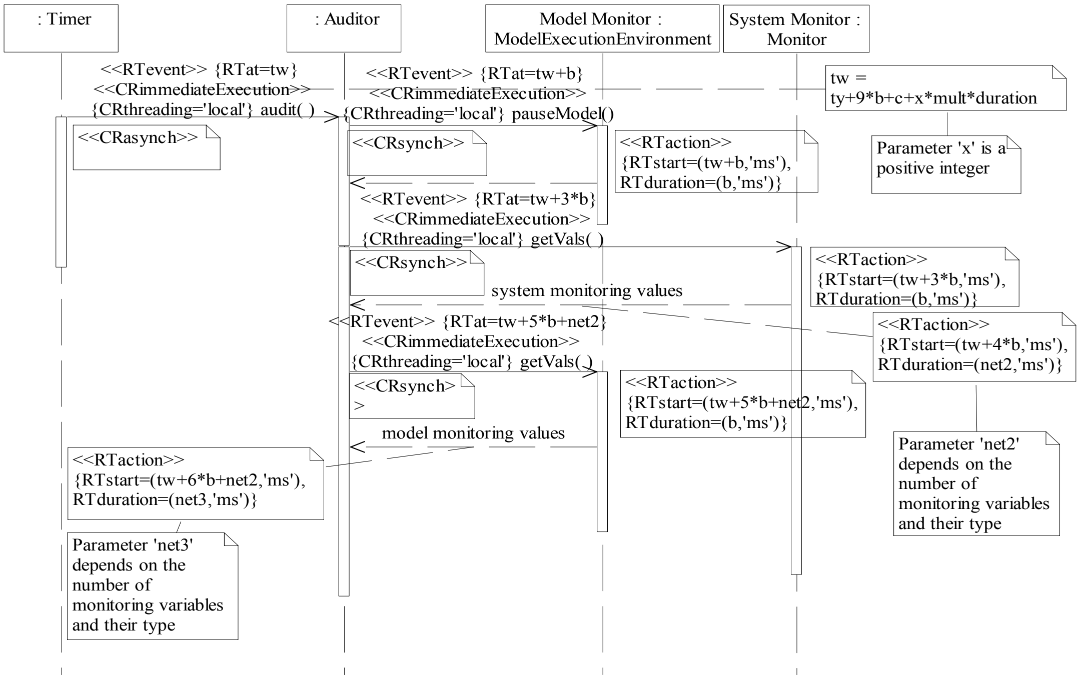
: Timer
: Auditor
Model Monitor : ModelExecutionEnvironment
System Monitor : Monitor
<<RTevent>> {RTat=tw}
<<CRimmediateExecution>>
{CRthreading='local'} audit( )
<<RTevent>> {RTat=tw+b}
<<CRimmediateExecution>>
{CRthreading='local'} pauseModel()
tw = ty+9*b+c+x*mult*duration
Parameter 'x' is a positive integer
<<CRasynch>>
<<CRsynch>>
<<RTaction>> {RTstart=(tw+b,'ms'), RTduration=(b,'ms')}
<<RTevent>> {RTat=tw+3*b}
<<CRimmediateExecution>>
{CRthreading='local'} getVals( )
<<CRsynch>>
system monitoring values
<<RTaction>> {RTstart=(tw+3*b,'ms'), RTduration=(b,'ms')}
<<RTevent>> {RTat=tw+5*b+net2}
<<CRimmediateExecution>>
{CRthreading='local'} getVals( )
<<RTaction>> {RTstart=(tw+4*b,'ms'), RTduration=(net2,'ms')}
<<CRsynch>>
<<RTaction>> {RTstart=(tw+5*b+net2,'ms'), RTduration=(b,'ms')}
model monitoring values
Parameter 'net2' depends on the number of monitoring variables and their type
<<RTaction>> {RTstart=(tw+6*b+net2,'ms'), RTduration=(net3,'ms')}
Parameter 'net3' depends on the number of monitoring variables and their type

Figure 5: Auditing Sequence Diagram

In this case, the result of the auditing algorithm is a remodeling indication. Otherwise the model is considered valid. Remodeling indication is constructed incrementally including all *OR* invalid conditions and the *AND* partial indication, if it exceeds the specified threshold. All of the above are denoted in detail in the activity diagram of figure 6, where the use of RTaction note stereotype enables the annotation of all action with estimation of their duration (tag RTduration). According to figure 6, the overall duration of the auditing is $14*b+g+net2+net3+k\ ms$, where *g* is the time for the audit tree to be built, *net2* and *net3* depend on the number and the type of the state monitoring variables, and *k* is upper bounded by $b+(aut.orNum+ aut.andNum)*(5*b+d)$.

Auditing execution are restricted by strict timing concerns, since the auditing tree must be constructed in a small fraction of the auditing 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 experiment designer is able to realize why 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.

4. IMPLEMENTING A SIMPLE FRT SIMULATOR

In order to evaluate FRTS RT-UML model correctness and flexibility, a simulator was built in Java based on class, activity and sequence diagrams corresponding to all FRTS activities as those discussed in section 4. FRTS model was constructed using RT-UML profile within the Rational Rose platform. The simulator was implemented taking advantage of automated code generation capabilities of Rose platform (program effort minimized). The system monitor environment and the simulation model execution environment are system specific and autonomously implemented.

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). Web site monitor and the simulation model were constructed in Java, as well. The web site can be modeled as a Multi-Queue, Multi-Server System. Visitor enquiries are classified into 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 $S_{i.}$ Denoting as d_i, the average queue delay, the following scenario was explored in the 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. The monitoring variables values must be collected by system monitor and also produced by the simulation model as output variables. Nine (9) discrete monitoring variables were defined. The number of Servers is one of them, as described in the following:

```
MV(3)   .mvname = servers
        .type = integer
        .ctechnique = integer
        .compParam = 1
        .indication = STRUCTURE
        .stateMonitoringIndication = TRUE
         rcondition.rcname = "server change"
```

Experimentation was conducted within Sun's Netbeans IDE and the Netbeans Profiler plugin. Table 1 presents corresponding results. Measured times (third column) are presented against estimated durations (second column) by the RT-UML FRTS model analysis. The table contains:

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 interval duration.

c) Audit duration.

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,	5004.09, 4871, 5278
Audit duration	5.479, 0.155, 123.441	2.700, 0.090, 53.400

Table 1: Basic FRTS time attribute comparison

Audit interval is explicitly defined. Since the basic operation duration (b) is not estimated, the measured time is used in audit duration estimation formulas.

For the estimation of 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 1, the following conclusions can be reached: a) audit interval is quite accurate, b) estimated audit duration is comparable to the measured one. Also, the estimation for maximum audit duration is higher than the measured one, indicating that the estimated maximum value may be used as the lower limit for audit duration.

5. CONCLUSIONS

The proposed RT-UML model for FRTS facilitates the exploration of timing issues and enables straightforward implementation of FRTS simulators, ensuring that all timing requirements are met. The detailed diagrams of FRTS activities specify how each of them implements its functionality in terms of events, activities, and actions, all of which have precise time orientation. Therefore FRTS experiment designer is able to infer estimations about time consistency and overall behavior of specific FRTS simulators. Time 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.

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] 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

[4] De Marco T., *Structured Analysis and System Specification*, Prentice Hall (Yourdon Press), Hemel Hempstead, 1978

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

[6] 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

[7] 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

[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] Anagnostopoulos D., M. Nikolaidou, "An Object-Oriented Modelling Approach For Dynamic Computer Network Simulation", to appear in *International Journal of Modelling and Simulation*

[10] 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.

[11] 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.

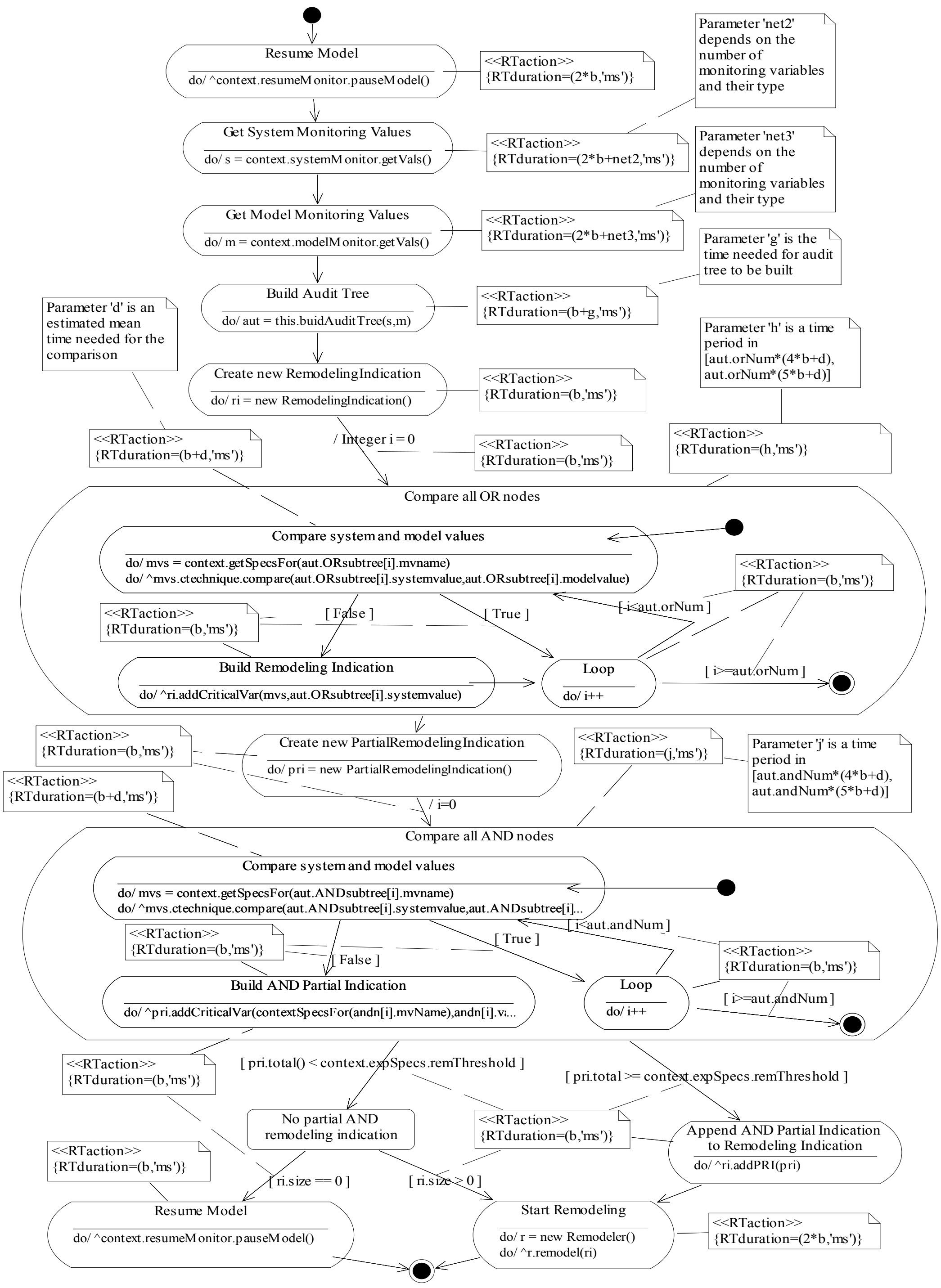
Resume Model
do/ ^context.resumeMonitor.pauseModel()
<<RTaction>>
{RTduration=(2*b,'ms')}
Parameter 'net2' depends on the number of monitoring variables and their type
Get System Monitoring Values
do/ s = context.systemMonitor.getVals()
<<RTaction>>
{RTduration=(2*b+net2,'ms')}
Parameter 'net3' depends on the number of monitoring variables and their type
Get Model Monitoring Values
do/ m = context.modelMonitor.getVals()
<<RTaction>>
{RTduration=(2*b+net3,'ms')}
Parameter 'g' is the time needed for audit tree to be built
Build Audit Tree
do/ aut = this.buidAuditTree(s,m)
<<RTaction>>
{RTduration=(b+g,'ms')}
Parameter 'd' is an estimated mean time needed for the comparison
Parameter 'h' is a time period in [aut.orNum*(4*b+d), aut.orNum*(5*b+d)]
Create new RemodelingIndication
do/ ri = new RemodelingIndication()
<<RTaction>>
{RTduration=(b,'ms')}
<<RTaction>>
{RTduration=(b+d,'ms')}
/ Integer i = 0
<<RTaction>>
{RTduration=(b,'ms')}
<<RTaction>>
{RTduration=(h,'ms')}
Compare all OR nodes
Compare system and model values
do/ mvs = context.getSpecsFor(aut.ORsubtree[i].mvname)
do/ ^mvs.ctechnique.compare(aut.ORsubtree[i].systemvalue,aut.ORsubtree[i].modelvalue)
<<RTaction>>
{RTduration=(b,'ms')}
<<RTaction>>
{RTduration=(b,'ms')}
[False]
[True]
[i<aut.orNum]
Build Remodeling Indication
do/ ^ri.addCriticalVar(mvs,aut.ORsubtree[i].systemvalue)
Loop
do/ i++
[i>=aut.orNum]
<<RTaction>>
{RTduration=(b,'ms')}
Create new PartialRemodelingIndication
do/ pri = new PartialRemodelingIndication()
<<RTaction>>
{RTduration=(j,'ms')}
Parameter 'j' is a time period in [aut.andNum*(4*b+d), aut.andNum*(5*b+d)]
<<RTaction>>
{RTduration=(b+d,'ms')}
/ i=0
Compare all AND nodes
Compare system and model values
do/ mvs = context.getSpecsFor(aut.ANDsubtree[i].mvname)
do/ ^mvs.ctechnique.compare(aut.ANDsubtree[i].systemvalue,aut.ANDsubtree[i]...
[i<aut.andNum]
<<RTaction>>
{RTduration=(b,'ms')}
[True]
[False]
<<RTaction>>
{RTduration=(b,'ms')}
Build AND Partial Indication
do/ ^pri.addCriticalVar(contextSpecsFor(andn[i].mvName),andn[i].va...
Loop
do/ i++
[i>=aut.andNum]
<<RTaction>>
{RTduration=(b,'ms')}
[pri.total() < context.expSpecs.remThreshold]
[pri.total >= context.expSpecs.remThreshold]
No partial AND remodeling indication
<<RTaction>>
{RTduration=(b,'ms')}
Append AND Partial Indication to Remodeling Indication
do/ ^ri.addPRI(pri)
<<RTaction>>
{RTduration=(b,'ms')}
[ri.size == 0]
[ri.size > 0]
Resume Model
do/ ^context.resumeMonitor.pauseModel()
Start Remodeling
do/ r = new Remodeler()
do/ ^r.remodel(ri)
<<RTaction>>
{RTduration=(2*b,'ms')}

Figure 6: Auditing Activity Diagram