AN RT-UML MODEL FOR BUILDING FASTER-THAN-REAL-TIME SIMULATORS

Dimosthenis Anagnostopoulos[1], Vassilis Dalakas[2],
Georgios-Dimitrios Kapos[1], Mara Nikolaidou[2]

[1]Harokopio University of Athens, 70 El. Venizelou Str, 17671, Athens, Greece
{dimosthe, gdkapos}@hua.gr
[2]University of Athens, Panepistimiopolis, 15771, Athens, Greece
{vdalakas, mara}@di.uoa.gr

ABSTRACT

Faster-than-real-time simulation (FRTS) is widely used for training, control and decision making purposes. FRTS experimentation proves to be rather demanding, requiring a consistent specification for developing such systems. This paper presents guidelines for an implementation framework, based on an industry standard, the Unified Modeling Language (UML). In particular, using the OMG UML Profile for Schedulability, Performance and Time Specification (abbreviated by Real-Time UML or RT-UML), specific timing attributes can be included in the derived UML model, which makes FRTS independent of the application examined. Thus, the implementation of relative program modules can be analyzed and realized, following the guidelines of this model, ensuring the reliability of the results within predetermined time frames.

KEYWORDS

Faster-than-Real-Time Simulation, RT-UML, Simulation Methodology, Systems Analysis.

1 INTRODUCTION

FRTS is used when attempting to reach conclusions for the behavior of a real system in the near future [3]. In this type of simulation, model execution is concurrent with the evolution of the real system. Thus, the advancement of simulation time must occur faster than real world time. Furthermore, FRTS implementation becomes more demanding, due to the hard requirements real time systems have for interacting with other agents [4].

In [1] a conceptual methodology for FRTS was described, aiming at providing a framework for conducting experiments dealing with the complexity and such 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, which involve its input data, operation parameters and structure. To deal with system modifications, remodeling adapts the model to the current system state. 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.

Experimentation phase thus comprises monitoring, that is, obtaining and storing system and the 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).

Specific measures are monitored to determine whether system reformations have occurred. The variables used to obtain the corresponding values are referred as monitoring 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.

To achieve a consistent transition from the analysis of FRTS systems to the implementation of the corresponding program modules, a detailed and multi-facet specification is provided here, using UML [5, 6]. The 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 RT-UML [7]. This profile, also used in [2], enables the detailed specification of critical time and synchronization requirements for FRTS components and the overall performance evaluation. Therefore, a detailed and integrated specification for FRTS systems is given, 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. Construction and execution of FRTS can now be performed assuring the validity of the results.

In section 2 we review 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. Due to the large extend of the detailed FRTS specification, section 4 provides only sample diagrams that specify how FRTS components implement their functionality in terms of events, activities, and actions, all of which have precise time orientation. Finally, in section 5, some conclusions are drawn.

2 RT-UML OVERVIEW

In UML, system modeling is based on different kinds of diagrams providing views of three main aspects of the system: structure, dynamic behavior, and management. In this paper, we define the structural and behavioral characteristics of FRTS with *use case*, *class*, *activity*, and *sequence diagrams*.

The UML diagrams just mentioned, do not provide the required degree of precision (regarding timing issues) for the specification of FRTS. Thus, we propose the use of RT-UML, 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. To give emphasis on time and concurrency aspects of FRTS systems, one is able to use elements only 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 [7]. Stereotypes applied to classes in class diagrams are displayed in the class box, above the name of the class (a in Figure 1). However, when tag values need to be specified for a certain stereotype, a *note* is also attached (b in Figure 1). In sequence diagrams, event stereotypes are displayed over the events, while method invocation and execution stereotypes are displayed in *notes* (c in Figure 1). In activity diagrams, *notes* are also used to indicate the application of a stereotype on an activity, state or transition (d in Figure 1).

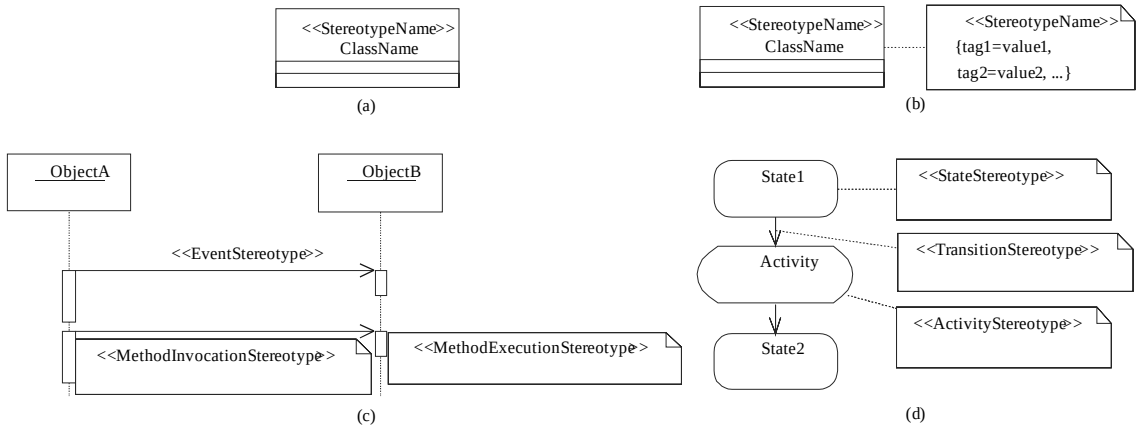

Figure 1. RT-UML notation

The RT-UML stereotypes used in the proposed FRTS model are briefly discussed here. 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. The method invoked when the object moves to "executing" state is specified with the *CRmain* tag. *RTtimer* models timer mechanisms. It defines two tags: tag *RTduration* specifies the time period after which the timer produces an event, while *RTperiodic* indicates whether the timer is periodic or not. *RTaction* is used in activity diagrams for methods, specifying the time instance they start (tag *RTstart*) and their duration (tag *RTduration*).

3 FRTS: A HIGH-LEVEL DESCRIPTION

This section is an overview of FRTS systems in terms of UML constructs. The use case diagram depicted in Figure 2 presents the entities involved in FRTS. Both the system and the model are separate from the main module of FRTS and can be viewed as distributed systems. System environment (SE) represents the actual system, as well as a surrounding mechanism which is responsible for performing monitoring of the real system. We consider it 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, defining the case study.

In particular, 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, respectively. How these values are collected and stored in both environments is not of our concern. We examine only the interchange of data. The 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).

The sequence diagram in Figure 3 emphasizes the communication between the entities described in the previous diagrams (User, FRTS System, ME and SE), in terms of message exchange. Initially, the user provides the experiment specifications (SetExperimentSpecifications) and starts the process. Thus, the FRTS System starts system monitoring (message to System Environment), initializes and starts the model, and starts model monitoring (messages to ME). In periodic, predefined time instances Audit (or state audit) is invoked. The model is then paused and the values of monitoring variables are retrieved from both the SE and ME (with GetSystemMonitoringInformation and GetModelMonitoring Information). Depending on the result of auditing the model is resumed (valid audit) or remodeling is performed and the old model is deleted (invalid audit).

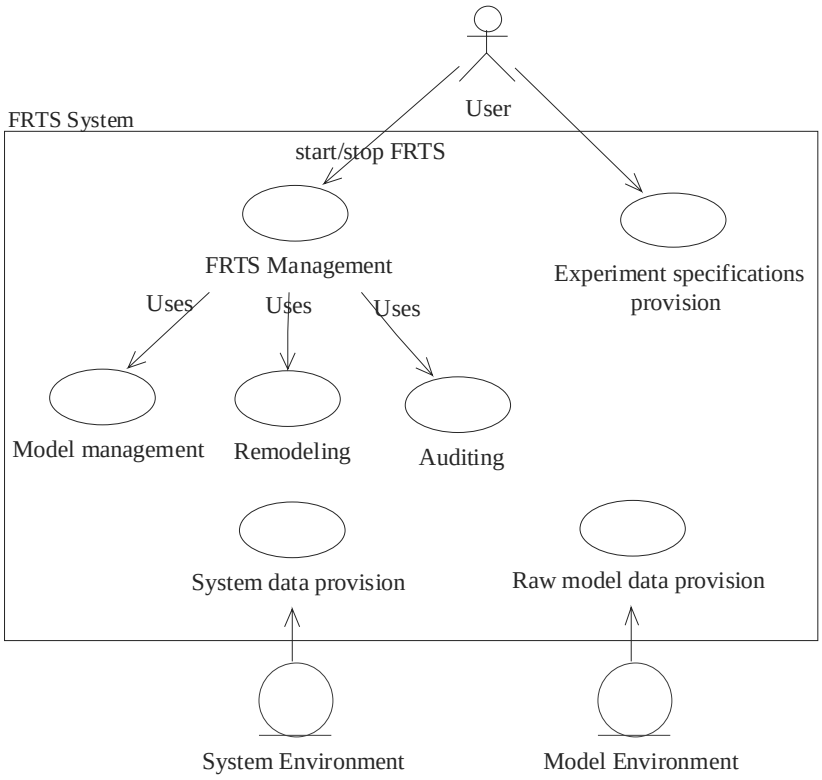

Figure 2. FRTS detailed use case diagram

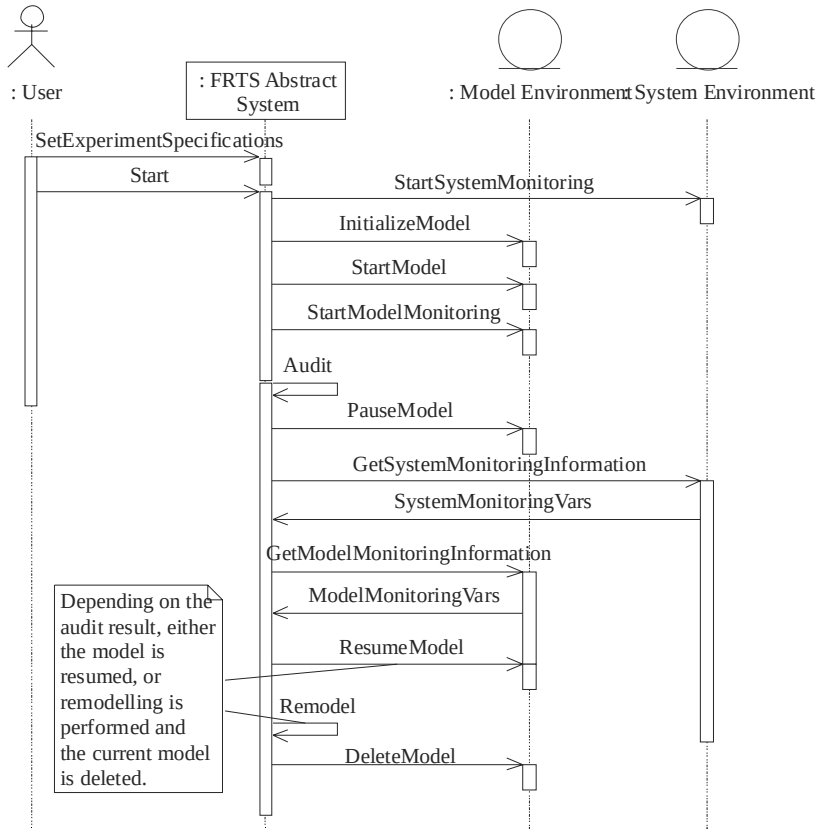

Figure 3. Main sequence diagram

4 FRTS System Specification

The FRTS system design is based on a set of classes (*Context*, *Control*, *Timer*, *StateAuditor*, *Auditor*, *Remodeler*, and *UserInterface*) and interfaces (*IAuditor*, *Monitor*, and *SystemMonitor*, *ModelExecutionEnvironment*), depicted in the class diagram of Figure 4. The *Context* is used for storing the experiment specifications, references to the system monitor and the model environment, and monitoring variable values used for state auditing. The *Control* initiates the FRTS process and the *Timer* is responsible for producing *StateAudit* and *Audit* events. *StateAuditor*, *Auditor*, and *Remodeller* are responsible for performing the homonymous operations. Both *StateAuditor* and *Auditor* classes implement the *IAuditor* interface. *Monitor* models the abstract concept of a variable values monitor, which is extended by interfaces *SystemMonitor* and *ModelExecutionEnvironment*. 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. Class *UserInterface* is simply the means for introducing user requests and data.

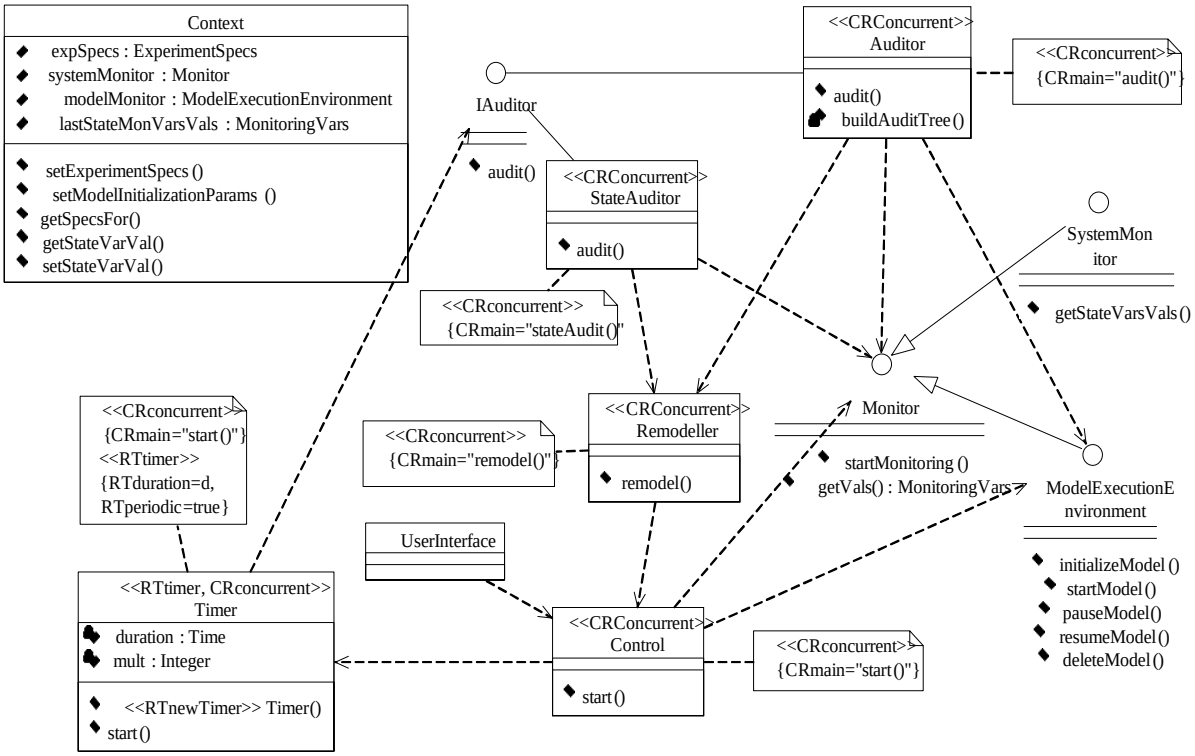

Figure 4. The main FRTS system classes

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 simultaneously. The *CRmain* tag of *CRconcurrent* stereotypes indicates the method that is executed when objects of each class are activated. Class *Timer* is a periodic producer of events, as indicated by the *RTtimer* stereotype.

The activity diagram of Figure 5 specifies the functionality of the *start()* method of *Control*. Each of the activities is annotated with the appropriate *RTaction* stereotype note. These are used to specify the duration of the activities. The lower part of each activity defines the actions executed (*do/*) or messages sent (*do/^*). Overall duration of method *start()* may be calculated as an amount of *6*b+c ms*, where *b* is the time needed for a basic operation to be performed (arithmetic operation, method invocation, etc.) and *c* is a parameter that depends on the specific FRTS application and the experiment specification. 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 the *Timer* has been started.

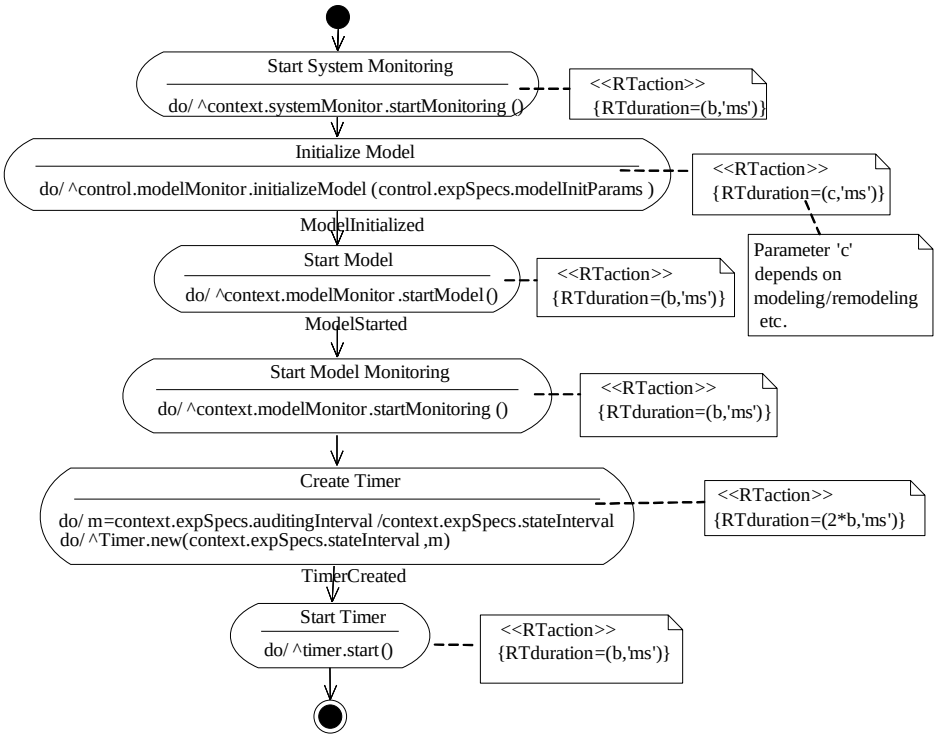

Figure 5. Activity diagram for method *start* of class *Control*

5 CONCLUSIONS

In FRTS model execution is concurrent with the evolution of the real system. Thus, FRTS implementation becomes more demanding, due to the hard requirements real time systems have for interacting with other agents. To achieve a consistent transition from the analysis of FRTS systems to the implementation of the corresponding program modules, a detailed and multi-facet RT-UML is provided in this paper. The descriptive capabilities of distinct types of 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. A detailed and integrated specification for FRTS systems is given, leading to standardized implementations of such systems that meet strict time requirements. Therefore, construction and execution of FRTS can be performed assuring the validity of the results.

REFERENCES

[1] Anagnostopoulos, D., Nikolaidou, M., & Georgiadis, P. (1999). A Conceptual Methodology for Conducting Faster Than Real Time Experiments. *Transactions of the Society for Computer Simulation International*, 16/2, 70–77.

[2] Bertolino, A., Marchetti, E., & Mirandola, R. (2002). Real-Time UML-based Performance Engineering to Aid Manager's Decision in Multi-project Planning, in: *The Proceedings of the Third International Workshop on Software and Performance (WOSP)*, (pp. 251–261), ACM Press, Rome.

[3] Cleveland, J. *et al.* (1997). Real Time Simulation User's Guide. NASA, Langley Research Center: Central Scientific Computing Complex.

[4] Fishwick, P., & Lee, K. (1999). OOPM/RT: A Multimodelling Methodology for Real-Time Simulation. *ACM Transactions on Modelling and Computer Simulation*, 9/2, 141–170.

[5] *OMG Unified Modeling Language Specification*, v1.5, on-line at http://www.omg.org/docs/formal/03-03-01.pdf

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

[7] *UML Profile for Schedulability, Performance, and Time Specification*, v1.0, on-line at http://www.omg.org/docs/formal/03-09-01.pdf