A Structured-Analysis-Based Specification for Faster-than-Real-Time Simulation

Dimosthenis Anagnostopoulos[1], Vassilis Dalakas[1], Mara Nikolaidou[1], and Vassilios Vescoukis[2]

***Abstract*—Despite the broad use of real-time and faster-than-real-time simulation (FRTS), especially for training and control purposes, formalisms focusing on experiment planning and execution have not yet been established. Having previously introduced a conceptual methodology for conducting FRTS experiments, we adopt a structured-analysis-based specification that is widely used in the real-time system domain and propose a consistent specification for developing FRTS systems. Orientation is towards the activities (processes) that have to be carried out and both data and event flows transferred from external agents to processes and vice versa, as well as inter-process flows. A common specification is presented for developing FRTS systems for diverse application domains. Due to the hard real-time constraints, timing issues related to the execution of simulation activities are also analytically examined. A case study where FRTS is applied on a real-time multi-teller system provides an in-depth analysis of the specification introduced.**

***Index Terms*— Simulation Methodology, Faster-than-Real-Time Simulation, Software Engineering, Structured Analysis**

I. 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 task, this issue has already been resolved [2], [3]. FRTS experimentation deals with the requirements imposed by the concurrent execution of the real world system and corresponding simulation model, thus it should be further explored. Real time systems often have hard real-time requirements for interacting with a human operator or other agents [2], consequently imposed to FRTS model execution [4]. Current FRTS research directions involve the distribution of the experiment over a network of workstations, intelligent control [5] and fault diagnosis [6], interactive dynamic simulation [7] and modeling formalisms [8].

In [9] a conceptual methodology for FRTS was proposed, aiming at providing a consistent framework for conducting FRTS experiments. The following simulation phases have been identified: *modeling, experimentation and remodeling*. Experimentation phase was emphasizing, studying the complexity and hard real-time requirements imposed by the concurrent execution and synchronization of the real world system and the simulation model. Experimentation phase is usually supported by Simulators especially built for a specific experiment. FRTS experimentation phase can be viewed as a real-time system itself,

This research was supported in part by Pythagoras program (MIS 89198) co-funded by the Greek Government (25%) and the European Union (75%).

[1]Dimosthenis Anagnostopoulos, Vassilis Dalakas, and Mara Nikolaidou are with the Harokopio University of Athens, 70, El. Venizelou Str., 17671 Athens, GREECE (e-mail: {dimosthe, vdalakas, mara}@hua.gr, corresponding author V. Dalakas, phone: +30-210-954-9344; fax: +30-210-951-4759).

[2]Vassilios Vescoukis, is with the Software Engineering Lab in National Technical University of Athens, 9, Iroon Polytexneiou, 15780 Athens, GREECE (e-mail: v.veskoukis@cs.ntua.gr).

thus is should be modeled and thoroughly studied, prior the implementation of the FRT Simulator supporting it.

Structured analysis and real-time system specification techniques [10], [11], have been used to typically describe FRTS experimentation phase. A specification of simulation activities and information exchange among them was provided, while emphasis was given in activity control and experimental state transition.

Considering timing issues and the nature of the tasks to be accomplished, experimentation and remodeling complexity is significant. This paper adopts the simulation-activity scheme of the methodology discussed in [9], and provides a consistent specification of FRTS, resolving simulation control, data exchange and timing issues, as no such approach has so far appeared in the literature. To accomplish this objective, we consider FRTS as a real-time system and use a respective specification method to provide the required process/data-oriented specification of FRTS.

The paper contribution is thus summarized in the following:

(1) specification of a consistent methodology that that can be widely applied for performing FRTS experiments, emphasizing simulation activity control, experiment states and state transition,

(2) specification of data exchange among simulation components,

(3) resolving timing issues of experimentation and remodeling activities.

Structured-analysis specification methods and diagramming techniques are widely adopted for formally describing real-time systems [11]. Not discussing in terms of specific applications, common specifications are established in this paper for developing FRTS systems in diverse application domains.

A brief introduction to the FRTS conceptual methodology emphasizing experimentation phase in presented in section 2. In section 3, we discuss the structured-analysis specification method used for real-time systems and then present the process/data-oriented specification of FRTS. This extends to the critical (in terms of the real-time constraints) simulation activities, that is, monitoring, auditing and remodeling. A simulation example presenting a FRTS application on a multi-teller system modeled as a GI/G/s queue is included in section 4, while conclusions reside in section 5.

II. FRTS METHODOLOGY

In [9] 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 [9]. Evidently, if conditions (a) or (b) are fulfilled, remodeling is invoked without examining condition (c) (figure 1).

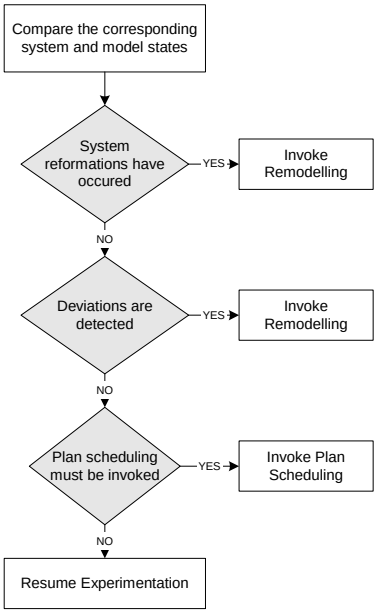

Fig. 1. Auditing control flow.

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 [9]. 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 [3], [12], or for domain/oriented approaches, such as computer networks [13]. 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 figure 2. 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.

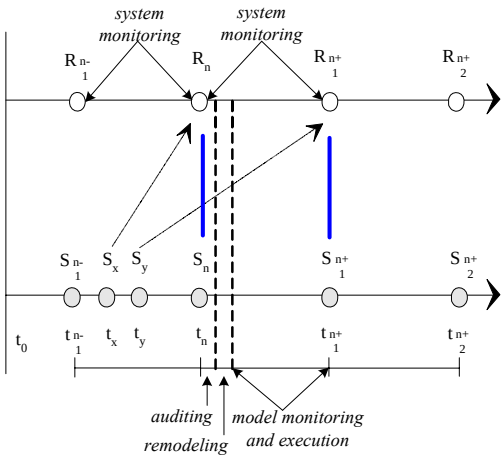

Fig. 2. Experimentation in FRTS

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.

Prior implementation, an 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. The whole idea is to have a formalized builded FRT Simulator to provide us the computationals limits in order to be able to choose a valid auditing interval.

III. FRTS Specification

A. Structured-Analysis Specification Method

Structured-analysis specification is widely used in real-time systems development [11]. We provide an overview of the major concepts and features of the specification method, which includes the following [11]: a) statement of purpose, b) environmental model (context diagram, event list), c) behavioral model (data flow diagrams, state transition diagrams, entity relationship diagrams) and d) the supporting textual specifications.

Data flow diagrams (DFD) model the basic system functionality in terms of the widely used constructs: *data transformations (processes), data flows and event flows*, *data stores* and *terminators,* as introduced by DeMarco and Yourdon [14], [15]. Diagramming techniques have been extended to support the real-time functionality. *Control processes* are used to control the invocation of data transformations, as discussed by Ward and Mellon [16], [17]. This is accomplished by event flows, depicted as dotted lines. There are two ways for process invocation: *E/D* denotes that a process is enabled and then disabled by the controlling process, while *T* denotes that a process is triggered and, when completed, it may return a corresponding signal. In the latter case, process execution duration is not determined by the controlling process. *State transition diagrams* (STD) show the dynamics of a system, i.e. how a system behaves over time and what causes the system to change its behaviour, in terms of s*tates*, *transitions*, *conditions* and *actions*. Finally, the supporting *project dictionary* [14], [15] holds the specification of all diagram components. In a widely-used notation for the specification of data flows [18], data composition is denoted by '+', multiple data elements by '{ }', choice of data elements by '[|]' and optional data elements by '( )'. The term 'elemental' denotes that data cannot be broken down any further. To maintain independence of specific application domains, we do not discuss E-R specification.

B. FRTS Activities

We use the above structured-analysis method to provide a process/data-oriented specification of FRTS. In the *context diagram* depicted in figure 3, the FRTS system is represented as a data transformation process (i.e. simulation activity). The event list includes the *start/ stop* experiment event flow. *System* and *model* terminators represent the actual system and the model and send *raw system data* and *raw model data*, respectively, to the FRTS system. When the model deviates from the current system state, remodeling is invoked and a *new model* is constructed to replace the old one. *User* terminator has the ability to start/stop the experiment. *Experiment specifications* consist of *monitoring variable, auditing interval, state interval, prediction interval* and *model initialisation* specifications, which are presented in table 1.

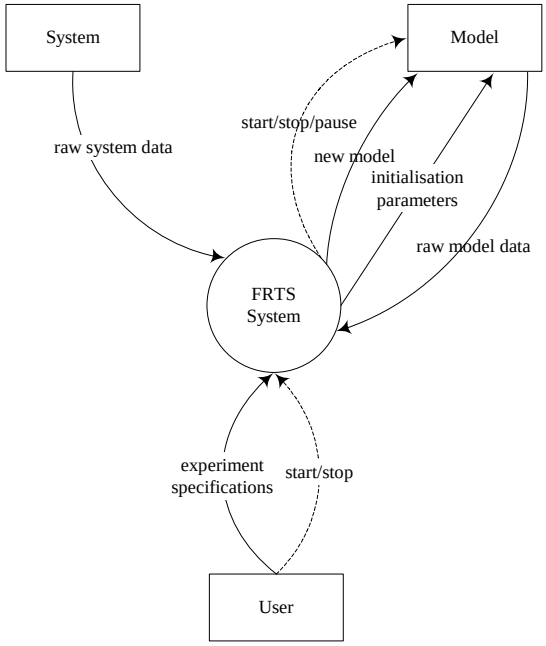

Fig. 3. FRTS context diagram.

TABLE I
EXTERNAL DATA FLOW SPECIFICATION

```
experiment specifications = monitoring variables specification +
                            auditing interval +
                            state interval +
                            prediction interval +
                            model initialisation parameters
  monitoring variables specification = {monitoring variable specification}
    monitoring variable specification = name +
                                        deviation range +
                                        state monitoring indication
      name = elemental
      deviation range = elemental /* determined for each experiment - real */
      state monitoring indication = [on | off] /* indicates if used in state monitoring */
  auditing interval = elemental
  state interval = elemental
  prediction interval = elemental
  model initialisation parameters = {model initialisation parameter}
    model initialisation parameter = model class +
                                     model parameters
      model class = elemental
      model parameters = {model parameter}
        model parameter = [model initialisation parameters | value ]
          value = elemental
raw model data = elemental /* model output data set */
raw system data = elemental  /* system output data set */
new model = elemental  /* new composite model */
```

The FRTS system (figure 3) is further decomposed in figure 4, where all FRTS functionality is illustrated. All the data flows shown in figure 3 are also depicted in figure 4, which is our main DFD. *Control Process* is responsible for controlling the experiment by examining event flows, which are set to either true or false, and invoking the corresponding activities. *Process Control* enables and disables *Get User Parameters*, *Initialise Model*, *Monitor Model*, *Monitor System*, *Audit* and *Remodel*. *Model* is initialized and executed according to experiment specifications. Monitoring of the model and the system is performed concurrently, and both types of raw data are collected. Data is then processed and respectively stored into *Model Data* and *System Data* stores. Monitoring is executed for a time period equal to auditing interval, such as $[t_{n-1}, t_n]$ in figure 2. Model

execution is then paused and *Audit* is invoked. *Monitor Model* is disabled during *Audit* and *Remodel*. *Monitor System*, though, is never terminated, so that system changes can always be perceived. System and model monitoring variables are calculated and then stored. *Audit* determines if the model still provides a valid representation of the system. If invalid, *Remodel* is invoked. The corresponding data flow specifications are presented in table 2.

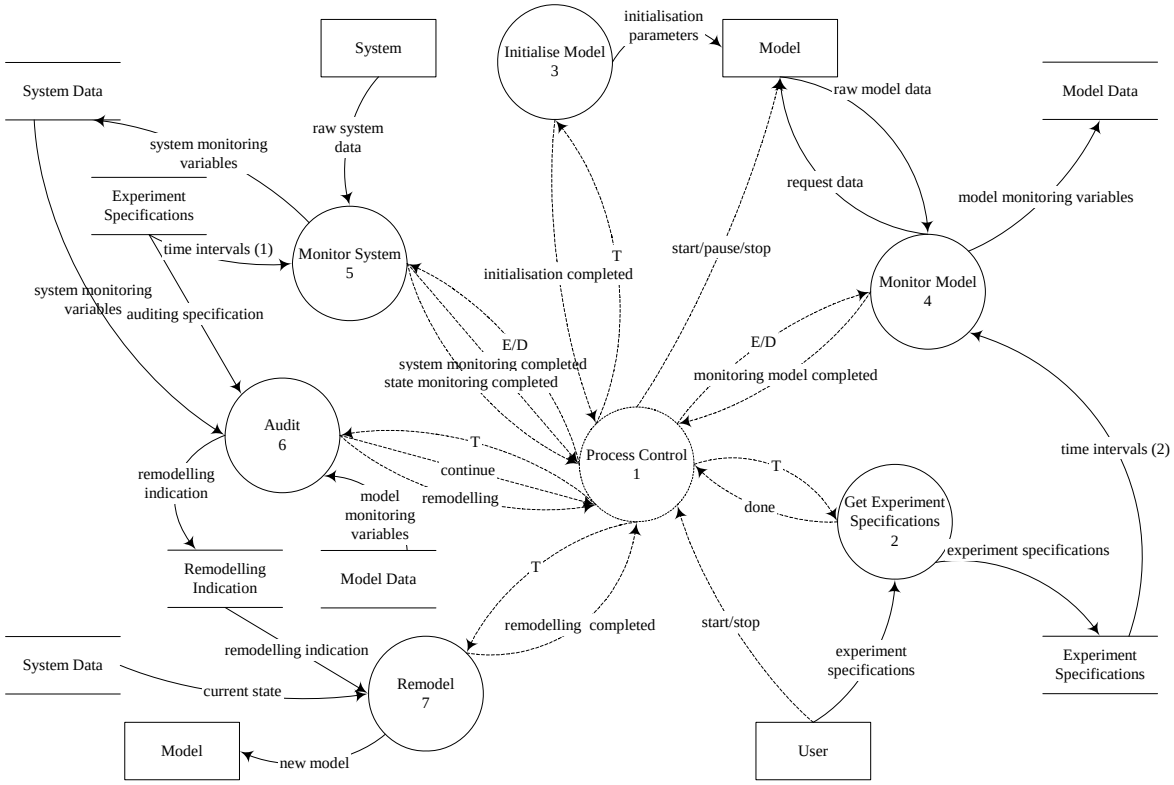

Fig. 4. FRTS activities, data and event flows.

TABLE II
FRTS DATA FLOW SPECIFICATIONS

```
time intervals (1) =   auditing interval +
                       state interval
time intervals (2) =   auditing interval +
                       prediction interval
model monitoring variables = {model monitoring variable}
model monitoring variable =  monitoring variable name +
                             output values
    monitoring variable name = elemental
    output values = {value}  /* stemming from multiple replications */
      value = elemental
system monitoring variables = {system monitoring variable}
system monitoring variable =  monitoring variable name +
                              output value
    monitoring variable name = elemental
    output value = elemental
auditing specification =  monitoring variables specification +
                          auditing interval +
                          state interval +
                          prediction interval
current state = system monitoring variables
remodeling indication = {indication type + variable names}
  indication type = [´structure´ | ´operation parameter´ | ´input data´ | ´deviation´]
    variable names = {name}
    name = elemental
```

Assume that the experiment is initiated (or re-initiated, after remodeling) at t_{n-1} and that we want to reach predictions for t_y

within the current auditing interval $[t_{n-1}, t_n]$ (figure 2). Thus, at t_{n-1}, $Sim(t_{n-1}) = t_{n-1}$. To achieve faster-than-real-time simulation, there has to be a point t_k, where $Sim(t_k) = t_y$, $t_k < t_n$. In this way, we define that the required *prediction interval* = $t_y - t_{n-1}$. Evidently, *prediction interval > auditing interval.* As discussed in [9], we usually choose a value p so that:

$$prediction\ interval = p \cdot auditing\ interval,\ \ p \in N^*$$

Model validation is performed through comparing the corresponding system and model states, described via monitoring variables, which are commonly defined for the model and the system. We emphasize the description of the corresponding data flows. Suppose that $MV_1, MV_2, \ldots MV_k$ are the monitoring variables. Each variable MV_i has two properties $MV_i.r$ and $MV_i.s$ for the system and the model, respectively. $MV_i.r$ 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 $MV_i.r = f_i(R_{i1}, R_{i2}, \ldots)$. $MV_i.s$ can also be calculated as a function of either a single-valued variate or an output stochastic process. As n replications are executed, $MV_i.s$ is calculated as a function of n stochastic processes. 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 $S_{ij} = g(S_{ij1}, S_{ij2}, \ldots, S_{ijk_j})$.

When comparing variables, we either use *one* or *n* values for model monitoring variables. In the first case, $MV_i.s = sum(S_{i1}, S_{i2}, \ldots, S_{in})/n$. In the latter, $MV_i.s$ is multi-valued. We describe monitoring variable specifications, which is part of experiment specifications forwarded to the FRTS system, when a comparison is performed with a single sample from the system and the model. Monitoring variable specifications, described in table 1, includes its *name*, a *deviation range*, which is determined for each specific experiment, and a *state monitoring indication.* State monitoring is described in the remainder of the paper. *Deviation range (dr)* supports a basic-inspection comparison between system and model data, that is, no deviation is encountered for MV_i when:

$$MV_i.s \in [MV_i.r \cdot (1 - dr),\ MV_i.r \cdot (1 + dr)]$$

The structure of monitoring variables – enabling realizing this comparison – is constructed combining $MV_i.r$ and $MV_i.s$ with the respective MV_i specification (table 1), where *name*, *state monitoring indication (smi)* and *deviation range(dr)* are user-specified. The structure of MV_i is the following:

MV_i = *(name, r, s, dr, smi)*

In the following paragraphs, we discuss the main FRTS activities, that is, *Monitor Model*, *Monitor System*, *Audit* and *Remodel*.

C. Monitoring

System monitoring (figure 5) collects *raw system data* and calculates *system monitoring variables*. When *Process Control* invokes *Monitor System*, monitoring of the real-time clock is also triggered. System monitoring never terminates, but is re-initiated whenever auditing is invoked (e.g. at t_{n-1}, t_n). Assume that function *realtime()* returns the current real-time point and that monitoring is re-initiated at t_n. *Monitor Real Time Clock* continues while:

$$realtime() - t_n < auditing\ interval - T_{Proc}$$

Then, *Control System Monitoring* activates *Determine System State,* which processes *raw system data* and calculates the values of *system monitoring variables* consuming T_{Proc}, which is considered as a constant throughout the experiment.

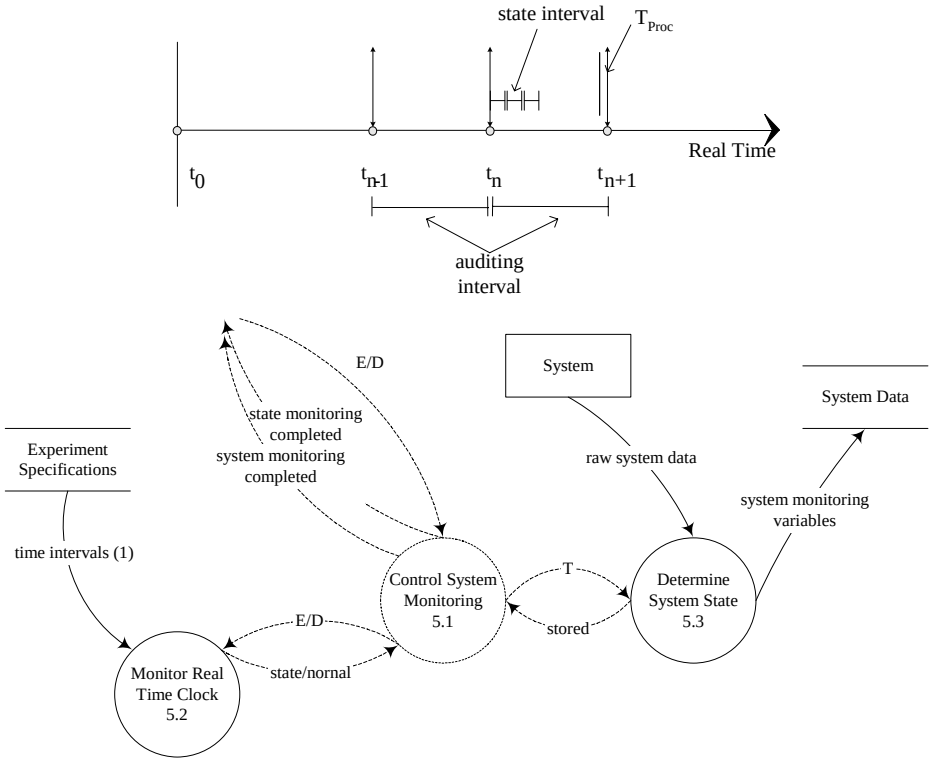

Fig. 5. Monitor System

In this way, handling deviations between the model and the system is performed after the auditing interval. This cannot be effective when critical, such as structural, modifications have occurred, where remodeling must be performed to restore consistency between the model and the system. In *state monitoring*, a limited set of system data is collected within a smaller interval *(state interval)* than the auditing interval, that is, while:

$$realtime() - t_{state\_init} < state\ interval$$

where t_{state_init} is initially set to t_n.Then, *state monitoring completed* signal is emitted, t_{state_init} is set to *realtime()* and state monitoring is re-initiated. State monitoring is efficient, as less computationally intensive, due to the amount of state monitoring data and the rather trivial comparisons it requires, so that $T_{Proc} = 0$ for calculating state monitoring variables. State monitoring variables are examined within *Audit* (*state auditing* activity) with no time overhead, as model execution is not paused. State interval duration is usually chosen so that:

$$auditing\ interval = g \cdot state\ interval,\ g \in N^*$$

Model monitoring (figure 6) is executed while the model is running, that is, while predictions are reached for the predetermined prediction interval within the given time frame (i.e. auditing interval). The predicted time point at any real time point is denoted by the simulation clock (figure 2). Assume that auditing is invoked at point t_n. Monitoring is initiated at t_x, after auditing and remodeling are completed, and simulation clock is set to the starting point of the current auditing interval, thus:

$$t_x - t_n = T_{Audit} + T_{Remodel}$$

$$Sim(t_x) = t_n$$

Monitor Simulation Clock duration is equal to the model execution time (T_{Exec}):

$$T_{Exec} = min \begin{cases} auditing\ \ inteval - T_{Audit} - T_{Remodel} - T_{Proc} \\ t_y - t_x, where\ Sim(t_y) = t_n + prediction\ \ interval \end{cases}$$

Then, *Control Model Monitoring* activates *Determine Model State,* which sends *request data* to the model (i.e. invokes the

corresponding method of the model object) and obtains *raw model data*. Data is then processed to calculate monitoring variables, consuming T_{Proc}. Then, model monitoring terminates.

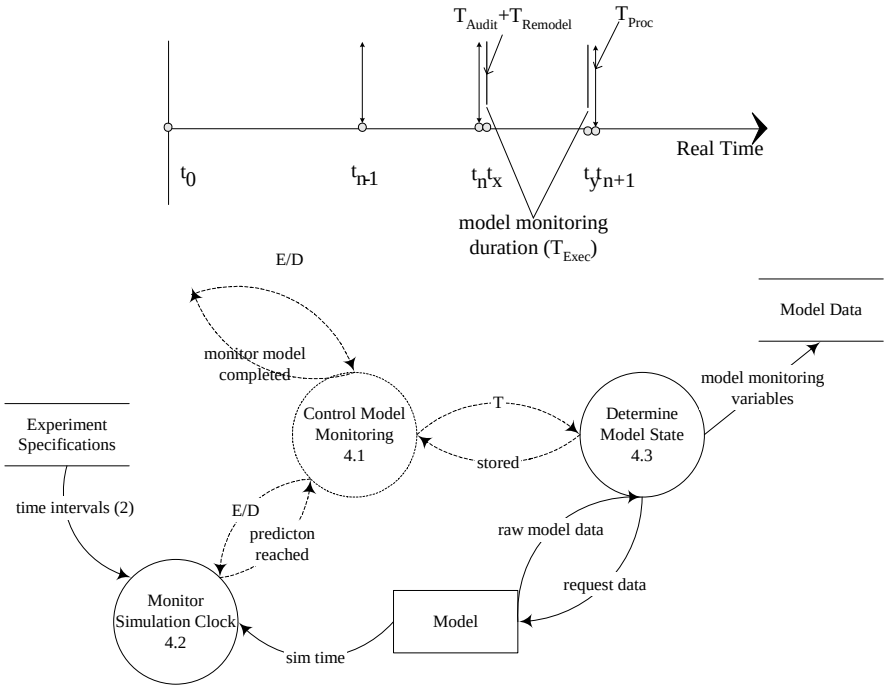

Fig. 6. Monitor Model

D. Auditing

Audit (figure 7) 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 *auditing interval*. Two distinct cases are thus considered: state auditing and standard auditing. Throughout this paper, the term *auditing* refers to standard auditing. State auditing is explicitly referenced.

1. State auditing: *Check System* 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 *remodeling indication* is produced. *Auditing Control* then notifies *Process Control*, so that remodeling is invoked. Only variables designated as *state monitoring variables* (table 1) are used in this process. As each variable may potentially cause remodeling, a potential state auditing algorithm (according to figure 7) is given in the following. This algorithm directly invokes remodeling to modify the model with minimum time overhead, without exhaustively examining all remodeling conditions.

```
for (i=1; i<=k; i++)
  if (MV_i.smi == on) && (deviates (MV_i.r, MV_i.s, MV_i.dr)) {
    build_remodel_indication(indication, MV_i.r, MV_i.s, MV_i.name);
```

```
    store(indication);
    return(modified);
  }
return(not_modified);
```

where $\texttt{deviates}(MV_i.r, MV_i.s, MV_i.dr) = \begin{cases} 1, MV_i.s \le MV_i.r \cdot (1 - dr) \text{ or } MV_i.s \ge MV_i.r \cdot (1 + dr) \\ 0, \text{otherwise} \end{cases}$

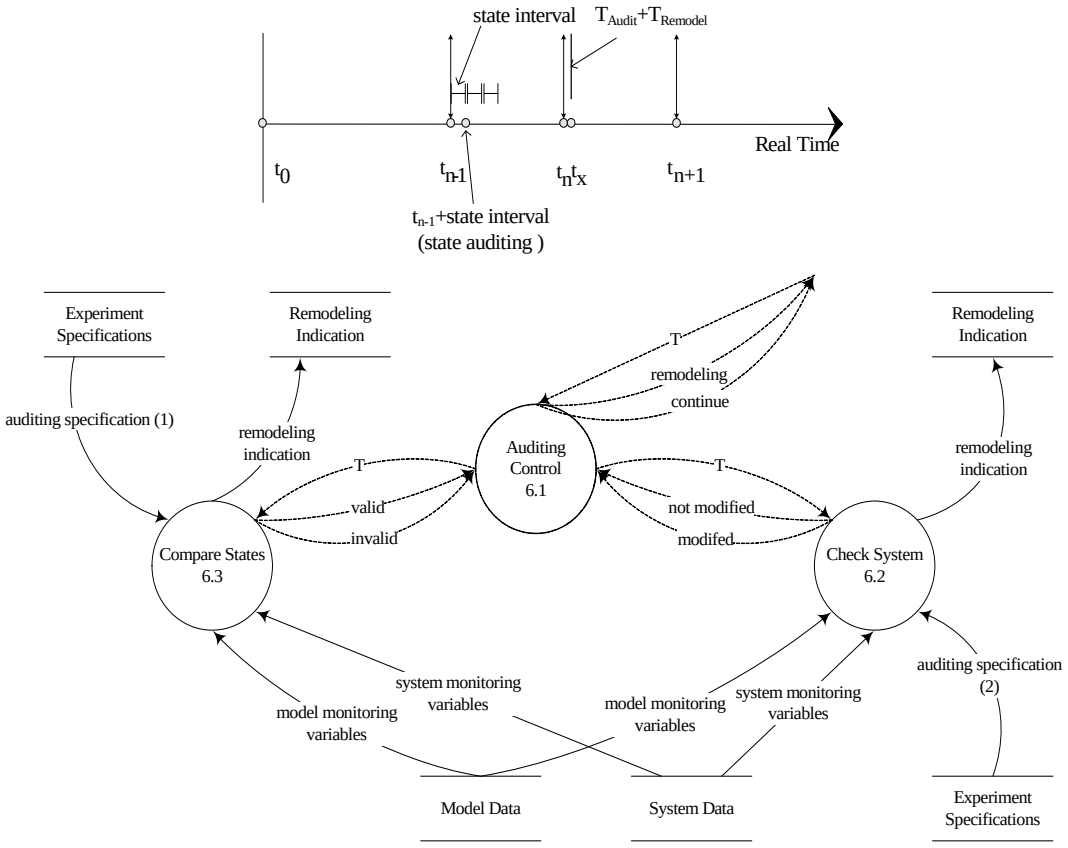

Fig. 7. Audit

2. Standard 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, *remodeling indication* is produced. *Compare States* examines the corresponding monitoring variables, and informs *Auditing Control*. All monitoring variables are used in this process. Considering that all comparisons may potentially cause remodeling, an algorithm determining whether remodeling should be invoked on the basis of the *k* monitoring variables is given in the following. This algorithm examines all potential conditions before invoking remodeling so that a complete indication is formed.

```
for (i=1;  i<=k;  i++)
```

```
    if (deviates (MVi.r, MVi.s, MVi.dr))
      build_remodel_indication(indication, MVi.r, MVi.s, MVi.name);

  if (!empty(indication)){
    store(indication)
    return(invalid);
  }
  else
    return(valid);
```

E. Remodelling

Remodel (figure 8) is invoked to handle system reformations and deviations between the system and the model. Especially for structural changes, accomplishing remodeling in real time is possible when model components are preconstructed and reside in model libraries, so that recompilation can be avoided [13].

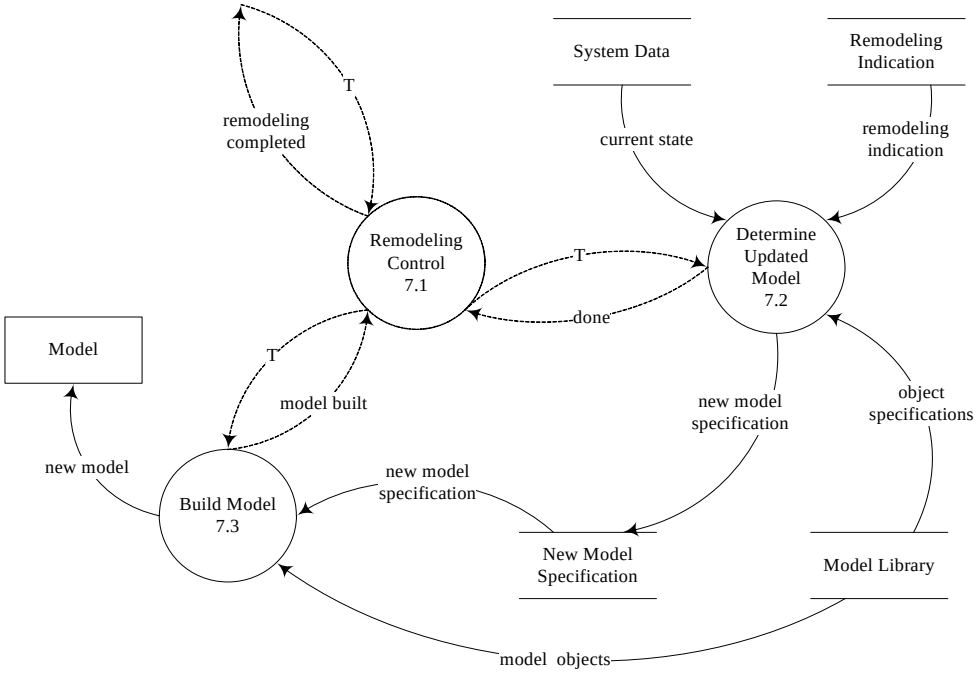

Fig. 8. Remodel

Determine Updated Model is the activity generating *new model specification* after examining *current state* (indicated by the system monitoring variables), *remodeling indication* and *object specifications*, which describe all available model classes residing in the library. The generated specification includes the new model class and model parameters, which involve its

properties and submodels. *Build Model* activity constructs and initializes the *new model* based on these parameters. When this is accomplished, *remodeling completed* signal is emitted back to FRTS *Control Process*. *Remodel* data flow specifications are presented in table 3.

F. FRTS State Transition

States and state-transition conditions are a substantial part of a discrete system description and are depicted in the STD of figure 9. There is a direct correspondence between an STD and a control process – whenever a control process exists on a DFD there is corresponding STD, and vice versa [10]. Conditions in the STD correspond to *incoming* event flows to the process and actions correspond to *outgoing* event flows, as depicted in figure 4 and figure 9. FRTS states concern only the activities that must be accomplished fulfilling the real time constraints, specifically *Model Initialising*, *Model Monitoring*, *System Monitoring*, *Audit* and *Remodel*. Apparently, there is also a one-to-one correspondence between STD states and FRTS processes, i.e., each simulation activity can be considered as a separate state.

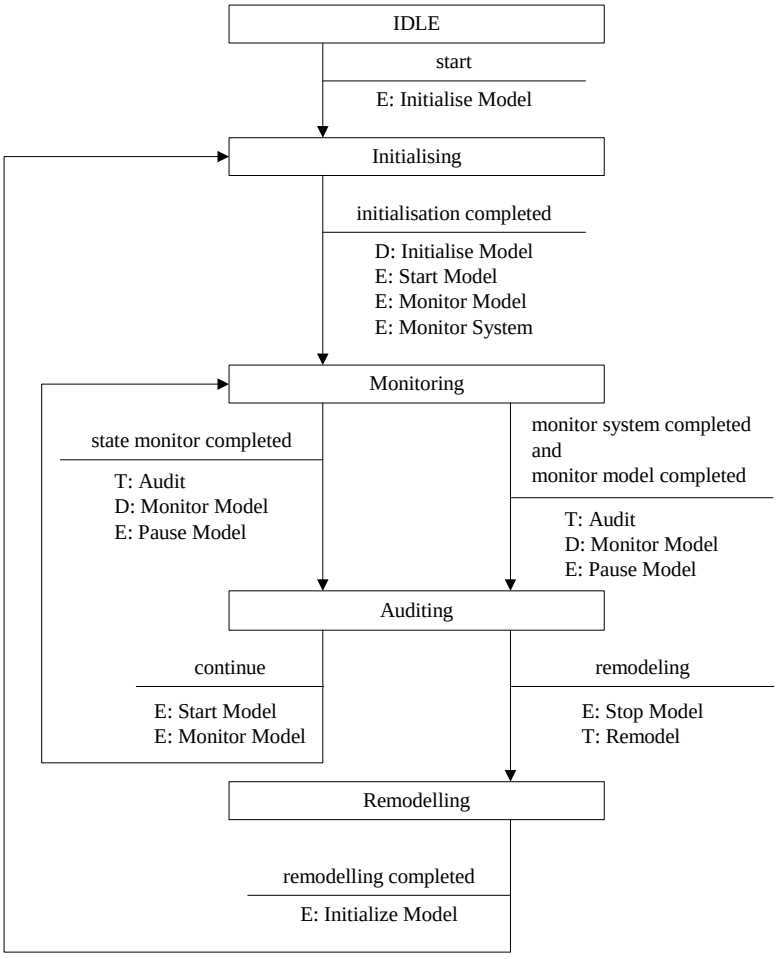

Fig. 9. FRTS state transition

TABLE III
REMODEL DATA FLOW SPECIFICATIONS

```
new model specification = new model class +
                                new model parameters
  new model class = elemental          /* object class, member of object hierarchy */
  new model parameters = {new model parameter}
    new model parameter = [new model specification | value]
      value = elemental
object specifications = {object specification}
  object specification =   model class +
                             model description
    model class = elemental
    model description = elemental      /* related to the model library organisation */
model objects = {model}
  model = elemental                    /* object class, member of object hierarchy */
```

IV. A FRTS EXPERIMENT OF A MULTI-TELLER SYSTEM

We describe a FRTS experiment for a multi-teller system modeled as a GI/G/s queue, where s≥1 (figure 10). This system has a variable number of servers that are modified during runtime, as servers may be abruptly activated or de-activated. The interarrival distribution type and parameters, as well as the service distribution type and parameters may also be modified, but all active servers have identical service properties. The objective of FRTS is to reach reliable conclusions for the near future and to ensure model validity taking into consideration potential system changes. The multi-teller example domain enables dealing with reformations of all three types. We discuss the potential reformations, deviations and main data flows: *monitoring variables*, *time intervals*, *model initialisation parameters, raw model and system data*, *remodeling indication* and *new model specifications* (figure 4).

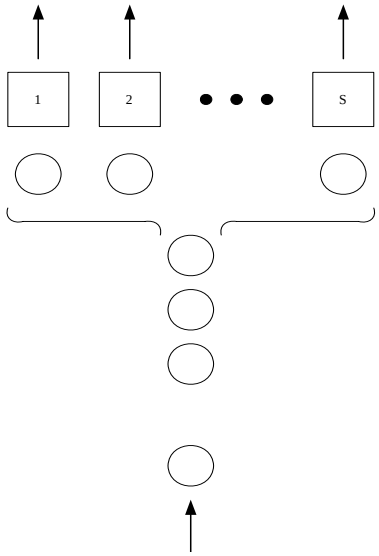

Fig. 10. GI/G/s system

In a GI/G/s system (queue), GI (general independent) refers to the distribution of interarrival times, G (general) refers to the distribution of service times and *s* is the number of servers. We use general distributions, as interarrival and service times may be

modified during runtime. Potential system reformations involve the following: the number of tellers, corresponding to a structural reformation, service distribution characteristics (type and parameters), corresponding to operation parameter reformations, and interarrival distribution characteristics (type and parameters), corresponding to input data reformations. We also consider two potential deviations, concerning the average time in the system and the average wait time in the queue, which may be detected even when reformations have not occurred. According to the above, we consider 7 corresponding monitoring variables: *teller_no*, *service_distr_type*, *service_distr_params*, *interarr_distr_type*, *interarr_distr_params*, *avgresponseD* and *avgwaitD*. Potential monitoring variable specifications (according to table 1) are the following:

1) *('teller_no', 0.0, on)*
2) *('service_distr_type', 0.0, off)*
3) *('service_distr_params', 0.2, off)*
4) *('interarr_distr_type', 0.0, off)*
5) *('interarr_distr_params', 0.2, off)*
6) *('avgresponseD', 0.3, off)*
7) *('avgwaitD', 0.3, off)*

As $MV_1.dr = 0.0$, no deviation is allowed between the number of tellers in the model and the system; otherwise, remodeling is invoked. $MV_1.smi = on$ denotes that this variable is used in state monitoring. *A deviation range = 0.0* is suitable for the service distribution type (MV_2). $MV_3.dr = 0.2$ denotes that a range of 20% is allowed between service distribution parameters. Considering a system for which auditing must be performed every 20 seconds, we set *auditing interval = 20.0* and *state interval = 2.0*, so that state monitoring is more frequent. To reach conclusions for 6 auditing intervals ahead, we set *prediction interval = 120.0*. Assume that a FRTS experiment starts at real time point 0.0 and that a M/M/1 model is initially used to represent the system. *Model initialisation parameters* determining the number of tellers and the distribution types and parameters of interarrival and service times are expressed as:

(MM1Obj, 1, (FIFOQueueObj1, exponential, 1.5), (ServerObj, exponential, 3.4))

The composite M/M/1 model consists of a queue model and a server model. The former acts as a customer generator and thus has the following properties: *interarrival distribution type = exponential* and *interarrival distribution parameters = 1.5 (λ=1.5)*. The teller model is respectively initialized.

The only variable used in state monitoring is *teller_no*. If $MV_1.r$ is not modified throughout the auditing interval, auditing will be executed at time point 20.0 As *prediction interval = 120.0,* model results and system data produced up to this point are the following:

Model		System
$MV_1.s$ .. $MV_7.s$ (20.0)	⟶	$MV_1.r$ .. $MV_7.r$ (20.0)
$MV_1.s$ .. $MV_7.s$ (40.0)		-
$MV_1.s$ .. $MV_7.s$ (60.0)		-
$MV_1.s$ .. $MV_7.s$ (80.0)		-
$MV_1.s$ .. $MV_7.s$ (100.0)		-
$MV_1.s$ .. $MV_7.s$ (120.0)		-

TABLE IV
MONITORING VARIABLE STRUCTURE AND REMODELING DECISION (AT 20.0)

MV_i	**Name**	**r**	**s**	**dr**	**smi**	**remodeling**
MV_1	*teller_no*	2	1	0.0	on	√
MV_2	*service_distr_type*	exponential	exponential	0.0	off	
MV_3	*service_distr_params*	0.29	0.34	0.2	off	
MV_4	*interarr_distr_type*	exponential	exponential	0.0	off	
MV_5	*interarr_distr_params*	0.23	0.28	0.2	off	√
MV_6	*avgresponseD*	0.49	0.55	0.3	off	
MV_7	*avgwaitD*	0.19	0.26	0.3	off	√

Assume that monitoring variables obtain the values illustrated in table 4 at point 20.0. Auditing indicates that a structure reformation has occurred based on MV_1. The remodeling indication formed is the following:

((‘structure’, ‘teller_no’), (‘operation parameter’, ‘interarr_distr_params’), (‘deviation’,’avgwaitD’))

Multi-teller systems are variable-structure systems consisting of queue and server entities, thus preconstructed and modular models have to be used to dynamically update the model. As $MV_1.s=2$, integration of an additional teller model is required. The model is directly imported from the model library and the new composite model is generated. In a similar way, if a teller were

de-activated, remodeling would remove and dispose the corresponding model component. Thus, *new model specification* is expressed as:

(MM2Obj, 2, (FIFOQueueObj,exponential,2.0), (ServerObj,exponential,3.1), (ServerObj,exponential,3.1))

When modifications are accomplished (also the ones imposed by MV_5 and MV_7), experimentation resumes from the current real time point. Assuming that 0.7 seconds were needed for auditing and remodeling ($T_{Audit} + T_{Remodel} = 0.7$), experimentation would resume at 20.7. As predi*ction interval = 120.0,* the model has to reach results for points 40.0 to 140.0 within auditing interval [20.0, 40.0], that is, before real-time point 40.0. There is always the possibility, though, that remodeling will be invoked during intermediate state monitoring points (e.g. 22.0, 24.0, etc) to update the model to critical system changes (i.e. teller activation and deactivation). Otherwise, auditing is executed at 40.0 to examine -once again- model validity.

V. Conclusions

We expressed the functionality of FRTS using structured-analysis, real-time system specification techniques to provide an analytical, formal description of this specific type of simulation. A conceptual FRTS methodology was used as a basis, as there is no formal approach focusing on experiment planning and execution. A process/data-oriented specification of simulation activities was provided, emphasizing activity control and experiment state transition, and respective timing issues were addressed. As simulation activities and control/data flows may be common in diverse FRTS implementations, we supported establishing a common basis for FRTS system development.

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] Gaafar L., "Maintaining the Validity of Simulation Models using Predictions Intervals", Computers and Industrial Engineering, Pergamon Press, vol. 37, 1999, pp. 859-871
[5] Cai Z., Y. Wang, J. Cai, " A Real-Time Expert Control System", Artificial Intelligence and Engineering, Elsevier Science, vol. 10, 1996, pp. 317-332
[6] 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
[7] Tyreus B.D., "Interactive Dynamic Simulation using Extrapolation Methods", Computer and Chemical Engineering, vol. 21, 1997, pp. 173-179, Pergamon Press
[8] Zeigler B. P., H. Praehofer, T. Kim, Theory of Modeling and Simulation (second edition), Academic Press, 2000
[9] 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
[10] Anagnostopoulos D., V. Dalakas, M. Nikolaidou, "Employing a Real-Time System Specification for the Development of FRTS Systems", Proceedings of SCS European Simulation Multiconference 2002 (ESM'02), Darmstadt, Germany, Society for Computer Simulation (SCS), June 2002.
[11] Goldsmith S., A Practical Guide to Real-Time Systems Development, Prentice Hall, 1993
[12] Barros F. J., "Modelling Formalisms for Dynamic Structure Systems", ACM Transactions on Modelling and Computer Simulation - TOMACS, 7(4), pp. 501-515, 1997
[13] Anagnostopoulos D., M. Nikolaidou, "An Object-Oriented Modelling Approach For Dynamic Computer Network Simulation", to appear in International Journal of Modelling and Simulation

[14] De Marco T., Structured Analysis and System Specification, Prentice Hall (Yourdon Press), Hemel Hempstead, 1978
[15] Yourdon E., Modern Structured Analysis, Prentice Hall (Yourdon Press), Hemel Hempstead, 1989
[16] Hatley D.J, I.A. Pirbahai, Strategies for Real-Time System Specification, Dorset House, New York, 1987
[17] Ward P.T., S.J. Mellon, Structured Development for Real-Time Systems, Prentice Hall (Yourdon Press), Hemel Hempstead, 1985
[18] Middelham F., “Traffic Simulation Predictability: Some Thoughts on Modelling”, Future Generation Computer Systems, Elsevier Science, vol. 17, 2001, pp. 627-636