Available at WWW.MATHEMATICSWEB.ORG POWERED BY SCIENCE DIRECT®

ELSEVIER Simulation Modelling Practice and Theory 10 (2002) 121–139

SIMULATION MODELLING PRACTICE AND THEORY

www.elsevier.com/locate/simpat

A methodological approach for model validation in faster than real-time simulation

Dimosthenis Anagnostopoulos

Department of Informatics, University of Athens, Panepistimiopolis, Athens 15771, Greece

Received 12 November 2001; received in revised form 1 March 2002

Abstract

When successfully dealing with time constraints, faster than real-time simulation (FRTS) improves process control capabilities through providing short-term predictions. Validation of a simulation model is accomplished through comparing system observations and model data corresponding to the same time points. A methodological approach is necessary for the realization of this comparison, considering the time-dynamic system behaviour and the potential deviations between the model and the system. A seven-step method is proposed, emphasizing the following issues: determining when predictions should be considered as valid, accomplishing validation on the basis of the available model and system data, considering that not all deviations between the model and the system are of equal significance, and indicating potential system changes, so that the model can be modified in real time. Computer networks are used as an example domain, due to their multi-entity structure and time-dynamic behaviour, offering excellent test cases for evaluating the proposed method. Experimental FRTS results from the application of the proposed method to the network domain are also presented.

Keywords: Faster than real-time simulation; Simulation methodology; Validation; Network simulation; Systems modelling

1. Introduction

Real-time simulation is widely used for the performance evaluation of systems behaviour in real time. Accomplishing real-time experimentation depends on the system speed and nature, which determines the allowed degree of human interaction with the system. When attempting to reach conclusions for the system behaviour

E-mail address: dimosthe@di.uoa.gr (D. Anagnostopoulos).

1569-190X/02/$ - see front matter
PII: S1569-190X(02)00091-6

in the near future, faster than real-time simulation (FRTS) is used, where advancement of simulation time occurs faster than real-world time. Making models run faster is the modeller's responsibility and a rather demanding task, as real-time systems often have hard requirements for interacting with the human operator or other agents. Relevant methodological issues have been discussed by Fishwick, who also proposed a multimodelling methodology based on the idea of selecting the appropriate model among multiple model types through trading structural information for faster runtime [1].

Validation is the process of determining whether a simulation model is an accurate representation of the system for the particular objectives of the study [2]. If the model is valid, it can be used to make decisions about the system. In FRTS, we have the unique capability to use system observations and model results both to test model validity and—in case of a valid model—to use simulation predictions to estimate future states of the system. This is based on the simple assumption that, if model validity can be consecutively ensured up to the current real-time point, it would be most probable that simulation predictions are also valid.

In this paper, a methodological approach for model validation in FRTS is presented, aimed at increasing our level of confidence for the predictions for a time-dynamic system under study. To achieve this, the model has to be validated in real time through obtaining and comparing system observations and model data. Despite the broad use of FRTS, there is no methodology describing how validation can be accomplished in FRTS experiments and, in particular, how this comparison should be realized. The closest work is that of Gaafa, who introduced a practical approach for maintaining the validity of simulation models during the simulation run, emphasizing the mathematical methods [3]. Provided that the system under study is observable (i.e. system observations can be obtained whenever required), the methodology proposed in this paper can be applied in diverse domains, which ensures its portability and scalability. Issues such as when the model should be considered as valid and how validation should be performed in order to indicate potential system changes are also addressed. Results from a prototype simulation experiment on a computer network are presented. As computer networks are multi-entity systems, they offer the capability to deal with the various types of structure and operation parameter changes that may occur. The issue of making the network model run faster than the system is not discussed in this paper, as achieving FRTS cannot be pre-assured. However, methodological approaches based on selecting the optimal model from a set of models employing a degree of abstraction are discussed in [1,4]. An alternative method suggesting the use of a single model type and multiple methods generated through approximation techniques is presented in [5].

In the following section, a conceptual methodology for conducting FRTS experiments is introduced and model validation issues are discussed. A methodological approach for model validation is presented in Section 3. Accomplishing validation according to the proposed method is examined in Section 4, in terms of a prototype application in the network domain. The simulation environment developed for this objective and experimental results are described in Section 5, while conclusions are presented in Section 6.

2. Faster than real-time simulation

When successfully dealing with time constraints, FRTS improves process control capabilities through providing short-term predictions. Experimentation is the most critical phase. A methodological framework has been proposed for conducting faster than real-time experiments, also considering the dynamic system behaviour [6]. An important category of system dynamics is structure variability. Variable structure models are thus used in simulation modelling, entailing in their description the possibility to change their own structure, i.e. their constitutive components as well as the relations that exist amongst them, corresponding to system changes [7]. Change of structure refers to the addition or deletion of single components [8], the change of interactions between components or the change of rules of behaviour [9]. Problems of model variation have been discussed within the context of object-oriented modelling, such as in the case of DEVS [10] and EMSY [11], mostly for control purposes.

The basic concept in the FRTS methodology is making dynamic systems serve as an information source, instead of adopting a doubtful statistical representation of their stochastic behaviour. The term *non-predetermined* or *time-dynamic systems* is used to denote actual systems that may be reformed during their evolution [6,12]. Reformations involve key features of a system, such as the system structure and operation parameters. Structure reformations involve not only the system components, but also the coupling relationships between them. *Structure* and *operation parameter reformation types* are thus distinguished.

The FRTS methodology consists of four phases: modelling, experimentation, remodelling and plan scheduling [6]. To compare the corresponding system and model states, both the system and the model are monitored during experimentation phase. System observations and model data are obtained within predetermined time intervals of equal length, called *auditing intervals*. In the case where the model states deviate from the corresponding system states, the model needs to be changed and thus remodelling is invoked. Deviations may be encountered due to system modifications, involving the system structure and operation parameters [6]. Remodelling modifies the model to depict the current system state. This is accomplished without terminating the real-time experiment, since no recompilation is performed. When model modifications are completed, experimentation resumes. Remodelling can also be invoked when deviations occur due to the stochastic nature of simulation, even when reformations have not been detected (i.e. system parameters/components have not been modified). In case none of the above has occurred and the simulation model is considered to be valid, the plan scheduling phase is invoked to take advantage of short-term predictions [6].

To conclude on system reformations, specific measures of the system and the model are monitored. The variables used to obtain the corresponding values are referred as *monitoring variables*. Monitoring variables should be considered at a conceptual level, as they do not follow the single-valued definition of program variables. Auditing examines monitoring variables corresponding to the same real-time points, and concludes on the validity of the model and predictions, as depicted in Fig. 1.

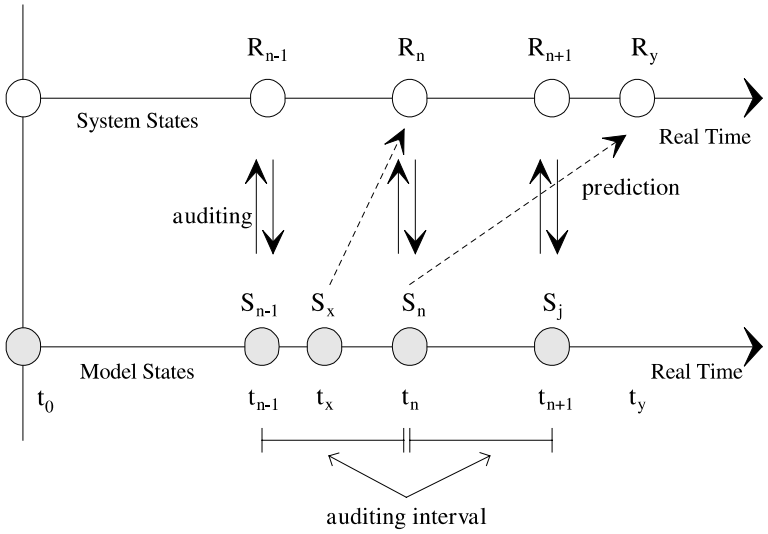

Fig. 1. Experimentation in faster than real-time simulation.

The evolution of the system and the model is depicted at the two horizontal axes. Real-time points are noted as t_i. The state of the system at point t_i is noted as R_i and the state of the model as S_i. When at time point t_x the model predicts the system state at point t_n (simulation time is equal to t_n), the notation $\mathrm{Sim}(t_x) = t_n$ is used. Thus, auditing process compares states S_x and R_n at time point t_n. If auditing detects that reformations or deviations have occurred, remodelling is invoked and the model composition scheme is modified [10]. When modifications are accomplished, the model is once more subjected to experimentation, starting from the current real-time point. Experimentation phase control flow in FRTS is depicted in Fig. 2. Timing issues concerning the execution of auditing and remodelling are discussed in [13]. As executing remodelling has an impact on the performance of simulation within this particular auditing interval, a method ensuring that FRTS is accomplished within the given time frame has also been introduced [13]. The performance of simulation within other auditing intervals is not affected.

The author argues that FRTS should be widely considered as a terminating simulation, which imposes that n independent replications are made. Each replication is terminated by a "natural" event that is scheduled for execution when simulation time reaches the predicted time points. Beginning with the same initial conditions, replications produce n observation sequences.

Suppose that $MV_1, MV_2, \ldots, MV_k$ are the monitoring variables used for the purposes of a FRTS experiment. Each variable MV_i is practically distinguished into two separate values $MV_i(r)$ 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

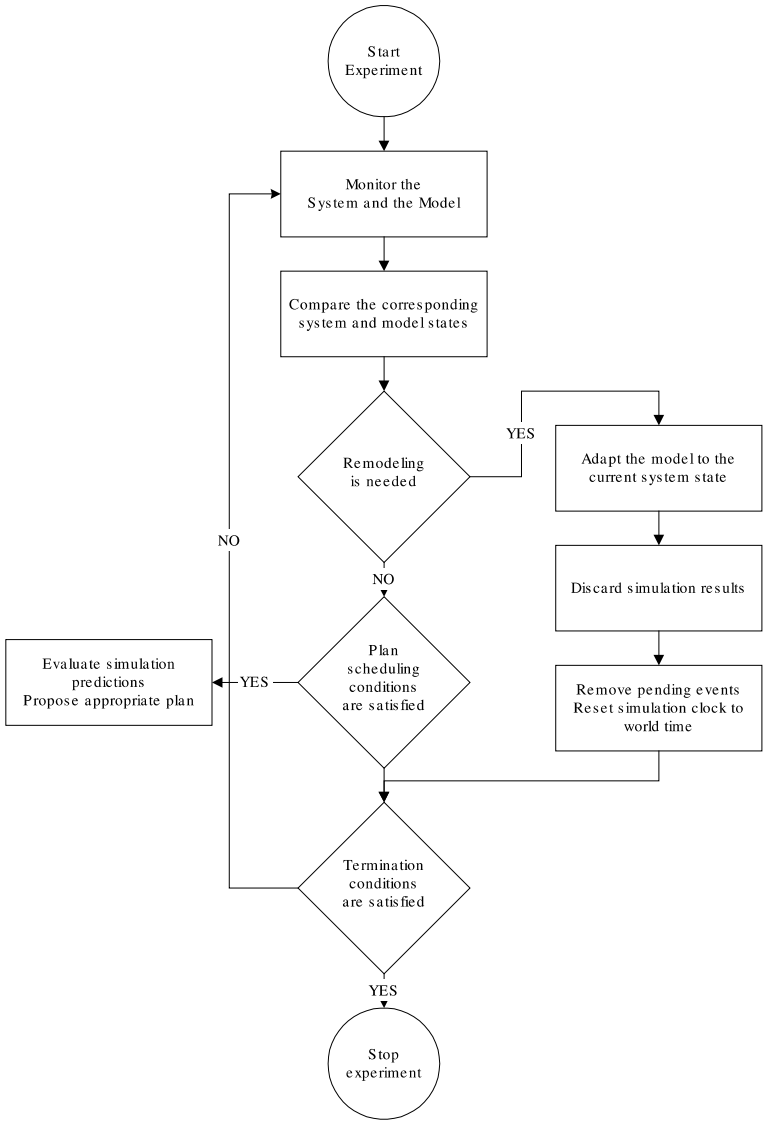

Fig. 2. Control flow of experimentation phase.

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 in the end 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, such as the number of customers that have been served in a GI/G/s system. Replication

results are thus extracted from $k_1, k_2, \ldots, k_n$ observations. Considering that the output process of each replication produces a single statistical sample S_{ij}

$$S_{ij} = g(S_{ij1}, S_{ij2}, \ldots, S_{ijk_j})$$

$$MV_i(s) = sum(S_{i1}, S_{i2}, \ldots, S_{in})/n$$

The issue of comparing system observations and simulation results has been thoroughly examined in the literature and various methods have been proposed depending on the nature of the problem. Law and Kelton provide an excellent review [2]. A confidence interval approach based on independent data proves to be a well-suited solution. As statistical methods have been widely discussed, this issue is not further emphasized. However, there is no simulation methodology focusing on model validation, accomplished through comparing FRTS results and system observations. Provision of a generic validation method is thus required to accomplish validation for diverse applicable domains on the basis of the available system and model data. The following requirements must be met to establish such a method:

1. Remodelling conditions must be expressed in a formal way in terms of system and model data. A deviation range must be allowed for each comparison between the corresponding data.
2. Model validity must be determined considering two types of remodelling conditions: the ones that—if fulfilled—autonomously cause remodelling, and the ones that are aggregately evaluated to decide whether remodelling is required. For the latter case, discrimination between conditions must be enabled, as certain conditions may be considered as more crucial than others.
3. Indication of the specific conditions causing remodelling must be enabled in order to incorporate system structure and operation parameter changes within the model.
4. A formal algorithm and appropriate data structures must be introduced for realizing the comparison between model and system data.

3. Validation method

The following seven-step method is proposed for realizing model validation in FRTS:

1. *Determining remodelling conditions:* involves distinguishing the two types of remodelling conditions: conditions that autonomously cause remodelling (*OR* conditions) and conditions that do not (*AND* conditions). Remodelling conditions specify whether the model should be considered as valid or invalid at a conceptual (i.e. descriptive) level.
2. *Determining monitoring variables:* involves the selection of the appropriate monitoring variables amongst the available system and model data.

3. *Expressing conditions through monitoring variables:* involves the conversion of remodelling conditions into well-formed expressions through the use of monitoring variables. A single condition may involve more than one monitoring variables and a single variable may be used by more than one condition. This step also includes the specification of the allowed deviation range for each comparison.
4. *Determining how much each condition contributes to the remodelling decision:* involves discriminating among the conditions that do not autonomously cause remodelling. As conditions are not equally significant, a weight factor must be assigned to each *AND* condition.
5. *Constructing the auditing tree:* the auditing tree is the structure maintaining system and model monitoring variables.
6. *Forming the auditing algorithm:* the auditing algorithm is automatically formed through accessing the auditing tree nodes, providing a low-level description of monitoring variable comparison.
7. *Executing the auditing algorithm through accessing the auditing tree.*

Selection of the deviation range depends on the nature of the experiment (i.e. how close should model states be to system states) and the specific method used to compare system observations and model data. For instance, in a basic-inspection-approach comparison, deviation range determines the lower and upper endpoints of the interval $[\mathrm{l}(MV_i(r)), \mathrm{u}(MV_i(r))]$ and the model is considered as valid when $MV_i(s) \in [\mathrm{l}(MV_i(r)), \mathrm{u}(MV_i(r))]$. When assigning weight factors to *AND* conditions, validation is performed using a scoring method [14]. Weights (or scores) are determined subjectively when conducting various aspects of the validation process and then combined to determine an overall score for the model. The model is considered as valid when this score is higher (or lower) than a threshold [15].

The *auditing algorithm* and *auditing tree* concepts are introduced for realizing the comparison between model and system data. The *auditing tree* (Fig. 3) is a dynamic structure. It is built whenever auditing is initiated according to the following specifications. When auditing terminates, the auditing tree is removed.

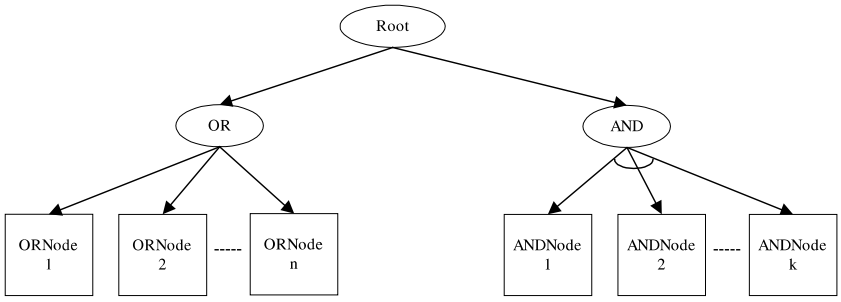

Fig. 3. Auditing tree structure.

1. The *auditing tree* is a conceptual tree structure (that is, it does not follow the formal definition of a tree). It is divided into two subtrees and includes two corresponding types of end nodes, *OR* and *AND*. End nodes of type *OR* represent conditions that autonomously—if fulfilled—cause remodelling. Nodes of type *AND* are aggregately evaluated to determine if remodelling is required. End nodes are directly accessed from *Root*—note that the two nodes named *OR* and *AND* in Fig. 3 are used to denote the corresponding subtrees and are not actually implemented. There are a_1 *OR* nodes and a_2 *AND* nodes, all of which are created as children of *Root*. In this way, $a_1 + a_2$ total accesses are required for all nodes. The number of tree nodes may be varying whenever auditing is performed.
2. Each end node corresponds to a single condition, which is expressed through the appropriate monitoring variables. However, a single condition may be expressed via more than one end node. End nodes are created and inserted in the appropriate subtree whenever an auditing tree is formed.
3. End node has the following structure: (*condition*, *system value*, *model value*, *deviation range*, [*weight*]) where *weight* field is used only for *AND* nodes, indicating the significance of each specific comparison for the remodelling decision. This structure corresponds to a basic-inspection-approach comparison between simulation data and real observations [2].

A code fragment for the implementation of auditing tree structures as Modsim III object classes is depicted in Fig. 4.

As nodes are directly accessed from *Root*, the total number of accesses required for tree nodes is $a_1 + a_2$. Accessing all nodes, we ensure that all remodelling conditions are evaluated prior to the initiation of remodelling and all reformations/deviations are detected, so that appropriate remodelling actions can be considered.

```
AudingTree = OBJECT;
        ORNodes = ARRAY [INTEGER] OF ORNode;
        ANDNodes = ARRAY [INTEGER] OF AndNode;
        ORNodeNum: INTEGER;
        ANDNodeNum: INTEGER;
END OBJECT;

ORNode = OBJECT;
        Condition: STRING;
        SystemValue: REAL;
        ModelValue: REAL;
        DeviationRange: REAL;
END OBJECT;

ANDNode = OBJECT;
        Condition: STRING;
        SystemValue: REAL;
        ModelValue: REAL;
        DeviationRange: REAL;
        Weight: REAL;
END OBJECT;
```

Fig. 4. Auditing tree implementation.

Upon completion of auditing, end nodes are removed. The auditing algorithm concludes that the model is invalid if at least one *OR* condition is fulfilled or the aggregate evaluation of *AND* conditions is fulfilled, that is

$$(C_1 = TRUE \ OR \ C_2 = TRUE \cdots OR \ C_{a_1} = TRUE \\ OR \text{ evaluation } (C_i, C_{ii}, \ldots, C_{a_2}) = TRUE),$$

where $C_1, C_2, \ldots, C_{a_1}$ are *OR* nodes and $C_i, C_{ii}, \ldots, C_{a_2}$ are *AND* nodes.

A sample auditing algorithm implementation is thus depicted in Fig. 5.

Another alternative that may be considered is to search the auditing tree for a single condition that may be fulfilled, and then invoke remodelling without accessing the overall tree structure. This would be acceptable if accomplishing remodelling does not impose that all remodelling conditions are previously detected, e.g. for reducing the time overhead. In this case, the auditing tree structure provides substantial capabilities for improving auditing performance, as the auditing algorithm would first search the *OR* subtree and then the *AND* subtree, while immediately terminate if a single remodelling condition was fulfilled. Depending on the nature of the experiment, the cost of such a search could be considerably less than the $a_1 + a_2$ accesses required in the general case. The auditing algorithm used also has an impact on remodelling, as remodelling becomes aware of either one or all fulfilled remodelling conditions. In the first case, remodelling rebuilds the overall model on the basis of the current system state. In the later, it is possible for remodelling to modify only the model features indicated by remodelling conditions. In this way, if conditions that were fulfilled only involve operation parameters, rebuilding the overall model structure would not be required, which results in a performance increase.

```
Root: AuditingTree;
NodeA: ORNode;
NodeB: AND Node;
...
FOR i:= 1 TO Root.OrNodeNum;
      NodeA:=Root.ORNodes[i];
      IF Deviates (NodeA.SystemValue, NodeA.ModelValue, NodeA.DeviationRange);
            Remodelling (NodeA.Condition);
      END IF;
END FOR;

FOR i:= 1 TO Root.ANDNodeNum
      NodeB:=Root.ANDNodes[i];
      IF Deviates (NodeB.SystemValue, NodeB.ModelValue, NodeB.DeviationRange);
            CalcWeight (TotalWeight, NodeB.Weight);
            BuildRemodelCondition (RemodelCondition, NodeB.Condition);
      END IF;
END FOR;

IF TotalWeight> Threshold
      Remodelling (RemodelCondition);
END IF;
```

Fig. 5. Auditing algorithm implementation.

4. A network FRTS example

Computer networks are used as an example domain for applying FRTS, as networks have a multi-entity structure and time-dynamic behaviour, offering excellent test cases for evaluating the proposed method. Potential variations that may occur in a time-dynamic system are discussed in [12], while a formal description of structure variability is provided in [7].

A discrete layering scheme of computer networks emphasizing communication-related issues is considered [16]. Network entities are protocols, communication and processing nodes, communication links and applications [17]. Network models consist of composite (coupled) and primitive (atomic) models. Applications and communication links are perceived as primitive entities. Network nodes are decomposed in terms of their communication elements (that is, the protocol stack) and network applications.

An object-oriented modelling scheme described in [18] is adopted. A network N consists of $n_1, n_2, \ldots, n_k$ processing nodes. The communication element is formed as a sequence of protocols $pr_1, pr_2, \ldots, pr_n$, starting from the lowest (MAC) layer. Since nodes are identical in communication aspects, network $N = \{n_i,\ 1 \leqslant i \leqslant k\}$ and $n_i = (pr_1, pr_2, \ldots, pr_n)$. The widely used TCP/IP stack along with an ethernet protocol could thus be represented as (10BaseT, IP, TCP). In this way, it is possible to build stacks representing the acceptable protocol combinations up to the highest supported layer. Applications can be viewed as sockets operating above the transport layer. Each node n_i is thus linked to a set of applications $A_i = \{a_{i1}, a_{i2}, \ldots, a_{ix_i}\}$, where x_i is the number of applications of n_i. The overall composition scheme of a computer network is given in Fig. 6.

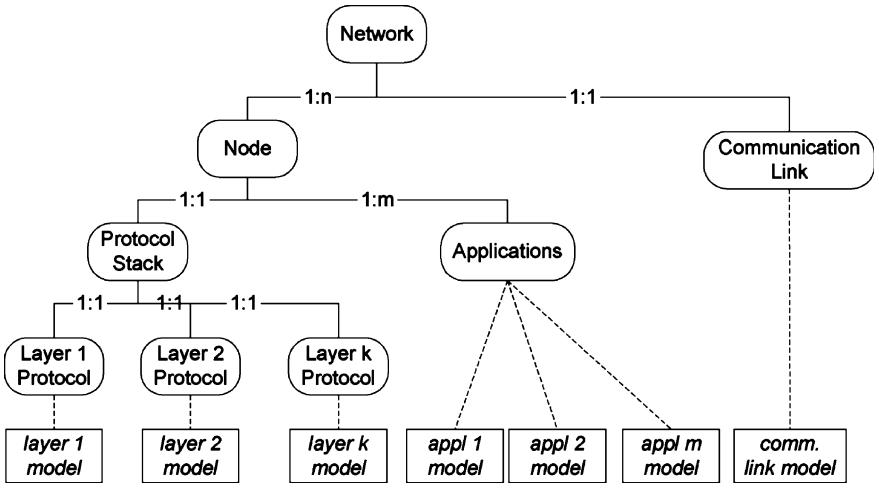

Fig. 6. Network composition tree.

The requirements for handling network modifications and ensuring model validity are supported through the use of modular models that have a hierarchical structure, according to which components are coupled together to form larger models [10]. This is accomplished using preconstructed model components, which are organized in object hierarchies and reside in model libraries. Preconstruction of primitive and composite models is enabled for all higher level entities, corresponding to the valid primitive entity combinations, and extends to the level where structural modifications may be encountered.

Network changes that must be considered involve the following structure and operation parameter changes: (a) node activation/deactivation (crash or shutdown), (b) initiation/termination of *critical* applications and (c) modification of application load. The term *critical* is used to denote applications having a considerable network load, as there are also applications of no practical impact. Node activation and deactivation, application initiation and termination are structure reformations. Other cases, such as application load modification, are operation parameter reformations. Depending on the reformation type, the model is appropriately modified during remodelling via either structure modifications or operation parameter value modification. To give an example, a network with four nodes and the corresponding applications is depicted in Fig. 7. The network is subjected to the following changes: $node_2$ and $node_4$ are deactivated and $node_5$ is activated. Applications of $node_2$ and $node_4$ are thus terminated. Note that other applications exchanging data with these nodes (i.e. $appl_{12}$) are also terminated.

The following symbolism is used to discuss the proposed method. Let $Snode = \{n_i,\ 1 \leqslant i \leqslant s\}$ be the set of nodes in the simulation model, and $Rnode = \{n_i,\ 1 \leqslant i \leqslant r\}$ the corresponding set in the network. Then, $node(S)$ and $node(R)$ denote the cardinality of *Snode* and *Rnode*, respectively. The set of critical applications operating on all model nodes is noted as *Sappl*, where an application is considered as *critical*—in the model or the network, respectively—when the number of data units (packets) or bits transmitted exceeds a predetermined threshold. The notation $a(S|R).field$ denotes an attribute of application a in the model or the network, respectively, e.g. $a(S).bits$. Also, $thrput(S).packets$ and $thrput(S).bits$ denote the aggregate number of packets/bits transmitted in the model, so that

$$thrput(S).packets = \sum_{i=1}^{s} \sum_{j=i+1}^{s} l_{ij}(S).packets, \quad thrput(S).bits = \sum_{i=1}^{s} \sum_{j=i+1}^{s} l_{ij}(S).bits,$$

l_{ij} denoting the load between nodes n_i *and* n_j. Then,

$$thrput(R).packets = \sum_{i=1}^{r} \sum_{j=i+1}^{r} l_{ij}(R).packets, \quad thrput(R).bits = \sum_{i=1}^{r} \sum_{j=i+1}^{r} l_{ij}(R).bits.$$

The proposed method is applied on the network example of Fig. 7. Remodelling conditions and the corresponding reformation types are presented in Table 1 (step 1). Remodelling conditions involve structure reformations, operation parameter reformations and deviations between the network and the model. In the general case, conditions of the same reformation type are not of equal significance. For instance, both

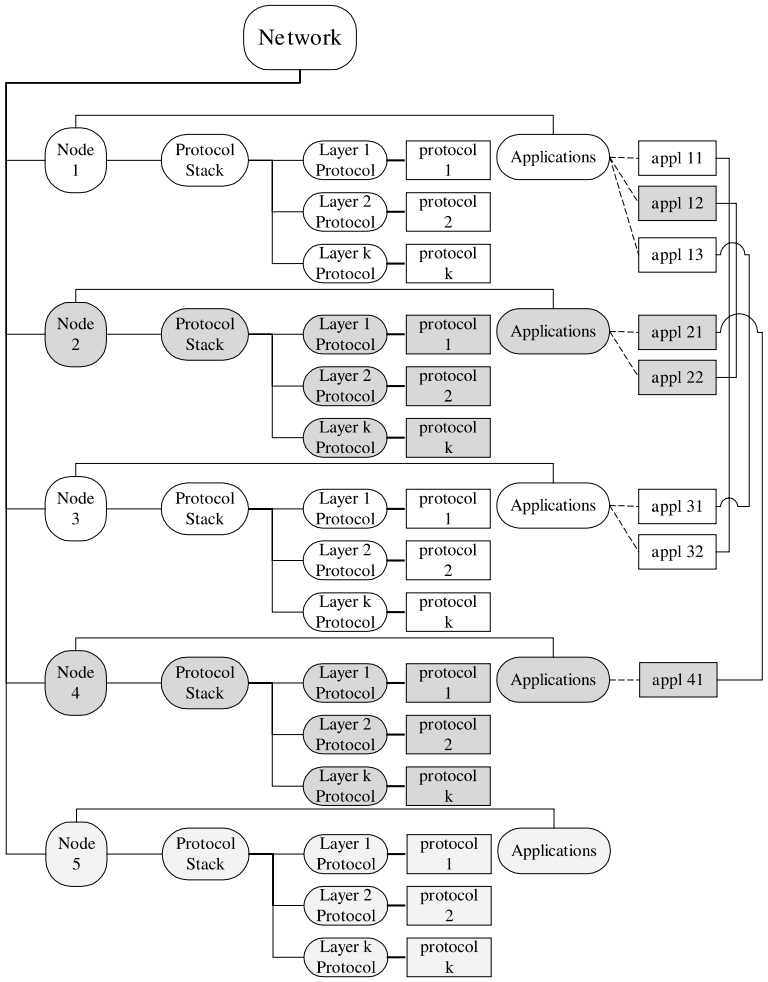

Fig. 7. Structure variation example.

Table 1
Remodelling conditions

	Condition	Reformation	Type	Weight
1	Node activation/deactivation	Structure reformation	*OR*	–
2	Application initiation/termination	Structure reformation	*AND*	w_a
3	Modification of application load	Operation parameter reformation	*OR*	–
4	Deviation in aggregate throughput	Deviation	*AND*	w_{t1}, w_{t2}

node activation and application termination are structure reformations, but only node activation is characterized as an *OR* condition. Monitoring variables are presented in Table 2 (step 2).

Table 2
Monitoring variables

Monitoring Variables		Representation
MV_1	Active nodes	$node(S\|R)$
MV_2	Critical applications	$appl(S\|R)$
MV_3	Packets transmitted by a	$a(S\|R).packets$
MV_4	Bits transmitted by a	$a(S\|R).bits$
MV_5	End-to-end delay of a	$a(S\|R).delay$
MV_6	Aggregate throughput (packets)	$thrput(S\|R).packets$
MV_7	Aggregate throughput (bits)	$thrput(S\|R).bits$

Table 3
Expressing remodelling conditions through monitoring variables

	Condition	Expression
1	Node activation/deactivation	$node(R)! = node(S)$
2	Application initiation/termination	$appl(R)! = appl(S)$
3	Critical modification of application load	$\forall a \in Sappl, Rappl$ 1. $deviates(a(R).bits, a(S).bits, d_{a1})$ 2. $deviates(a(R).packets, a(S).packets, d_{a2})$
4	Deviation in aggregate throughput	1. $deviates(thrput(R).packets, thrput(S).packets, d_{t1})$ 2. $deviates(thrput(R).bits, thrput(S).bits, d_{t2})$

Remodelling conditions are expressed using monitoring variables as depicted in Table 3 (step 3). To examine condition 3, for instance, real observations and simulation results are compared, concerning the packets transferred through application a, noted as $a(S).packets$ and $a(R).packets$, respectively. The *deviation range* is noted as d_a. Using the basic inspection method to implement this comparison, condition 3 is fulfilled when $deviates(a(R).packets, a(S).packets, d_a) = TRUE \iff a(S).packets \notin [a(R).\ packets(1 - d_a), a(R).packets(1 + d_a)]$.

The allowed deviation range varies according to the orientation of the simulation experiment. Lower values contribute to the reliability of simulation predictions but lead more frequently to remodelling, especially for *OR* nodes, which causes results to be discarded. High values have opposite effects. The lowest limit (0.0) is used when no deviation is acceptable, as for the number of nodes. In Table 3, conditions 1 and 2 are expressed differently than the others to indicate that deviation range is equal to 0.0. Weight values assigned to *AND* conditions are depicted in the last column of Table 1 (step 4).

An instance of the auditing tree is presented in Fig. 8 (step 5). This tree is formed after the structure variation depicted in Fig. 7. There are only two common applications in the network and the model, and thus four *OR* nodes are needed to examine application load modification. Deviation in the aggregate throughput (condition 4) is also represented as two *AND* nodes.

Remodelling is invoked according to the auditing algorithm presented in Fig. 9, which is automatically formed according to this auditing tree (step 6). Finally,

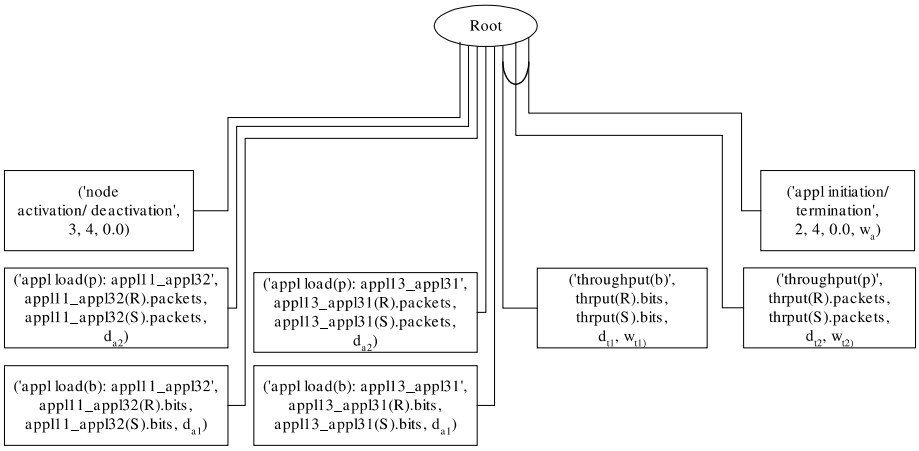

Fig. 8. Auditing tree instance.

```
IF  deviates (node(R), node(S), 0.0)
        Remodelling("node activation/ deactivation");
IF  deviates (appl11_appl32(R).bits, appl11_appl32(S).bits, da1)
        Remodelling("appl load(b): appl11_appl32");
IF  deviates (appl13_appl31(R).bits, appl13_appl31(S).bits, da1)
        Remodelling("appl load(b): appl13_appl31");
IF  deviates (appl11_appl32(R).packets, appl11_appl32(S).packets, da2)
        Remodelling("appl load(p): appl11_appl32");
IF  deviates (appl13_appl31(R).packets, appl13_appl31(S).packets, da2)
        Remodelling("appl load(p): appl13_appl31");
IF  deviates (appl(R), appl(S), 0.0) {
        CalcWeight(TotalWeight, wa);
        BuildRemodelCondition(RemodelCondition, "appl initiation/ termination"); }
IF  deviates(thrput(R).bits, thrput(S).bits, dt1) {
        CalcWeight(TotalWeight, wt1);
        BuildRemodelCondition(RemodelCondition, "throughput(b)"); }
IF  deviates(thrput(R).packets, thrput(S).packets, dt2) {
        CalcWeight(TotalWeight, wt2);
        BuildRemodelCondition(RemodelCondition, "throughput(p)"); }
IF  TotalWeight > Threshold
        Remodelling (RemodelCondition);
```

Fig. 9. Auditing algorithm of the network example.

at step 7, the auditing algorithm is executed through accessing the auditing tree nodes.

5. Simulation environment and results

A prototype FRTS experiment for a local network environment was performed. The implementation domain was the TCP/IP local network of a university campus building, consisting of 10Base segments. The simulation environment architecture

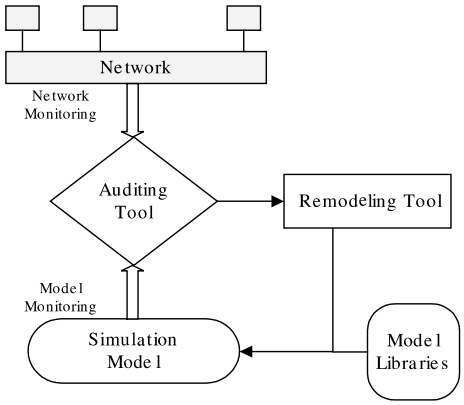

Fig. 10. Simulation environment architecture.

is illustrated in Fig. 10. Connections between modules indicate information flows and module invocations. Network monitoring is accomplished through SunNet Manager NMS. Network monitoring and auditing tools are implemented in C. Remodelling tool is integrated within the modelling platform to enhance interaction with the model. Simulation is executed on a single workstation. Monitoring results are forwarded to the FRTS environment when the auditing interval elapses.

A significant issue that must be addressed in FRTS is the length of the auditing interval. A short interval results in the following disadvantages: (a) frequent auditing, before a considerable amount of data is collected and (b) wasting time, since execution of auditing—and potentially remodelling—has a time overhead and no other activity can be carried out in parallel. The selection of a 10 s interval falls in this category. On the other hand, if a long interval were used, the simulation environment would not be aware of important events, such as a node crash, for a considerable time period. The selection of a 5 min monitoring interval would fall in this category. Based on a number of initial trials to ensure the feasibility of the experiment, a 60 s auditing interval was used.

Sample simulation results to indicate the nature of a FRTS experiment are presented. In Fig. 11, the number of applications in the model and the network are plotted for 17 consecutive monitoring intervals. In model validation, remodelling is often caused, as marked with the dark rectangles, due to the variation in the number of active sessions, even within neighbouring intervals (note that only application remodelling cases are marked). In time points 184 and 308, there are 40 and 33 active sessions, respectively, while during the intermediate interval only seven active sessions seem to exchange data. Whenever this occurs, models of applications that are no longer active are removed and new application models are initialised and integrated within the aggregate network model. Experimentation then resumes.

Model and system aggregate throughput is plotted in Fig. 12 for 19 consecutive monitoring intervals. During the first 10 monitoring intervals, variation of the actual throughput is relatively low. However, low deviation parameter values were used,

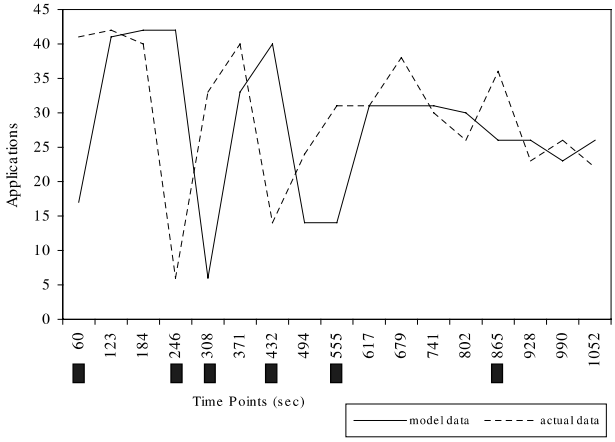

Fig. 11. Application number variation.

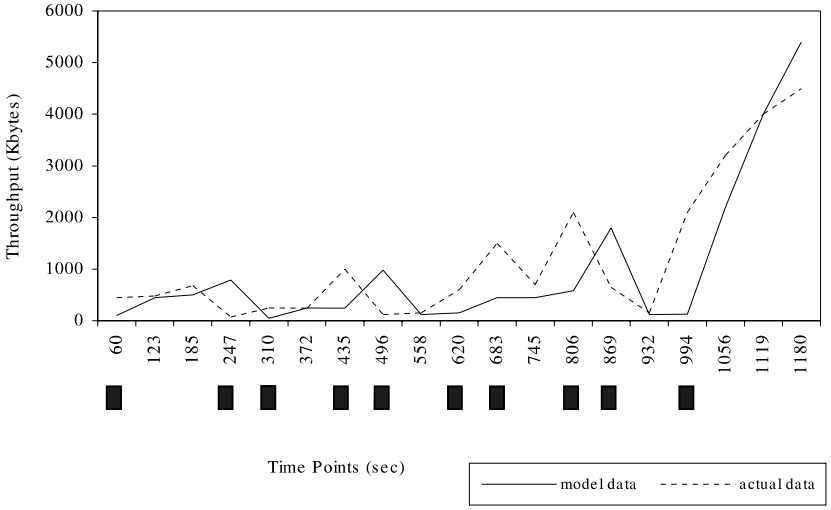

Fig. 12. Throughput variation.

forcing the model to be frequently subjected to remodelling, as indicated with the dark rectangles. During the late intervals, where traffic is increased, higher values were used. Simulation results were compared against network data to validate the model within consecutive auditing intervals. When the model repeatedly proves to be valid, short-term predictions were also assumed to be reliable. At point 1180, for instance, simulation results indicate that a bottleneck might occur, due to the load caused by a specific application, within the consecutive two intervals (an aggregate throughput of 12,000 KB is expected at time point 1299). In this case, *plan*

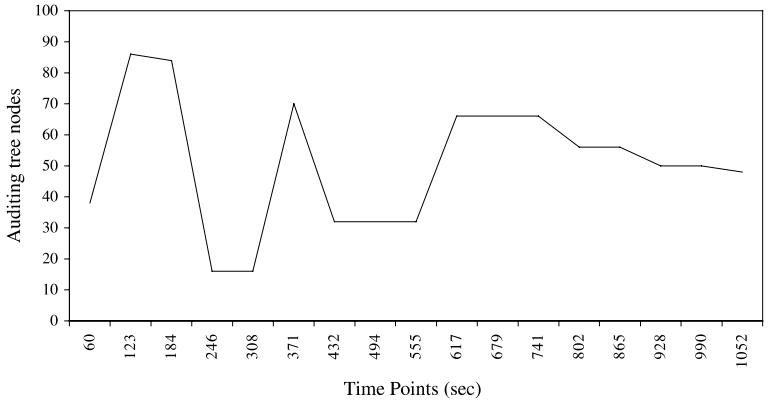

Fig. 13. Auditing tree end nodes.

scheduling is invoked to take advantage of predictions and propose an appropriate plan. This plan may be employed at the network management level, as simulation has no administrative authority on the system under study. Simulation focuses on reaching faster than real-time results and ensuring the validity of the model, but does not consider further actions after predictions are forwarded to plan scheduling.

In the experiments presented in Fig. 11 and Fig. 12, remodelling was caused due to application initiation/termination and deviations in the aggregate throughput. Even though the number of active nodes was not modified, the time-dynamic network behaviour resulted in numerous remodelling cases. Even after remodelling, where simulation is re-initiated, short-term predictions were reached for at least two auditing intervals ahead of real time. In model validation, the number of auditing tree nodes was modified according to the varying number of applications. For instance, considering that in Fig. 11 the number of common applications in the model and the network ranged from to 6 to 41, the tree had 16 end nodes at point 308 and 86 end nodes at point 123. Variation in the number of auditing tree end nodes for this experiment is presented in Fig. 13. Throughout all FRTS experiments, the number of auditing tree end nodes ranged from 6 to 136.

FRTS was exercised in two different ways. In the first case, depicted in Fig. 11 and Fig. 12, auditing searched the entire tree to detect all remodelling conditions. The time required to accomplish both auditing and remodelling varied according to the number of tree nodes, but was always less than 2 s, which was acceptable compared to the 60 s auditing interval. In the second case, the auditing tree was searched for a single condition that could be fulfilled, starting from the *OR* subtree. Remodelling was then invoked, without accessing the overall tree structure. In this case, remodelling was not aware of all remodelling conditions, but was invoked with a considerable lower time overhead, which could be crucial for specific application domains. In the above example, remodelling was invoked after searching less than half of the *OR* nodes, and the time overhead was respectively reduced to less than 1 s.

6. Conclusions

A seven-step method for comparing real observations and model data was introduced, being capable of dealing with the time-dynamic system behaviour. Creation of systems models that change their components and the interactions between these components along with their decision rules is strongly related to organisation learning, which is a significant research area in social sciences. Validation of such models is greatly enhanced when it is possible to compare model results against real observations. As model validity must be determined on the basis of multiple output variates, which are not of equal significance, a formal algorithm and appropriate structures were introduced for realizing this comparison. The applicability of the proposed method was examined using the network application domain. Based on a number of different FRTS experiments, it was concluded that, when remodelling can be invoked without detecting all conditions causing result invalidity, as in the case where taking advantage of reliable predictions should be instantaneous, validation is accomplished—using the auditing tree structure—far more efficiently. Portability of this method to diverse domains is ensured as long as the system under study is observable. Transportation, manufacturing and process-control systems are considered as potential application domains. The issue of taking advantage of reliable predictions was not considered in this paper, as it is strongly depended on the specific application domain of the FRTS experiment.

References

[1] P. Fishwick, OOPM/RT: a multimodelling methodology for real-time simulation, ACM Transactions on Modelling and Computer Simulation 9 (2) (1999).
[2] A.M. Law, W.D. Kelton, Simulation Modelling and Analysis, McGraw-Hill, 2000.
[3] L. Gaafar, Maintaining the validity of simulation models using predictions intervals, in: Computers and Industrial Engineering, vol. 37, Pergamon Press, 1999, pp. 859–871.
[4] R. Cubert, P.A. Fishwick, OOPM: an object-oriented multimodelling and simulation application framework, Simulation 70 (6) (1998) 379–395.
[5] A.J. Garvey, V.R. Lesser, Design-to-time real-time scheduling, IEEE Transactions on Systems, Man and Cybernetics 23 (6) (1993) 1401–1502.
[6] D. Anagnostopoulos, M. Nikolaidou, P. Georgiadis, A conceptual methodology for conducting faster than real-time experiments, SCS Transactions on Computer Simulation 16 (2) (1999) 70–77.
[7] F.J. Barros, Modelling formalisms for dynamic structure systems, ACM Transactions on Modelling and Computer Simulation 7 (4) (1997) 501–515.
[8] F. Cellier, H. Elmqvist, Automated formula manipulation supports object-oriented continuous system modelling, IEEE Control Systems 28 (2) (1993) 28–39.
[9] G. Lorenz, A. Urhmacher, K. Simon, H. Bossel, Application of artificial intelligence methods to the representation and modelling of tree growth, in: Proceedings of IUFRO: Artificial Intelligence and Growth Models for Forest Management, Vienna, Austria, 1989, pp. 121–130.
[10] B.P. Zeigler, Object-Oriented Simulation with Hierarchical Modular Models, Academic Press, 1990 (copyright by Author in 1995).
[11] A. Urhmacher, EMSY—an extended modelling system, in: Artificial Intelligence Expert Systems and Symbolic Computing for Scientific Computation, North Holland, Amsterdam, 1992, pp. 323–332.
[12] T. Pawletta, An object oriented framework for modelling and simulation of variable structure systems, in: Proceedings of SCSC '96, Portland, 1996, pp. 8–13.

[13] D. Anagnostopoulos, Experiment scheduling in faster than real-time simulation, in: Proceedings of PADS '02, IEEE Computer Press, Washington, 2002, pp. 163–171.
[14] O. Balci, How to assess the acceptability and credibility of simulation results, in: Proceedings of WSC '89, IEEE Computer Press, 1989, pp. 62–71.
[15] R. Sargent, Verification, validation and accreditation of simulation models, in: Proceedings of WSC '00, IEEE Computer Press, 2000, pp. 50–58.
[16] S. Jones, C. Smythe, A generic framework for the simulation analysis of protocol layered communication systems, Simulation Series 25 (1) (1993) 183–186.
[17] J. Cramer, J. Magee, Configuring distributed systems, in: Proceedings of 5th ACM Workshop on Models and Systems for Distributed Systems Structuring, Mont St. Michel, France, 1992, pp. 237–245.
[18] D. Anagnostopoulos, M. Nikolaidou, An object-oriented modelling approach for dynamic computer network simulation, International Journal of Modelling and Simulation 21 (4) (2001) 249–256.