Data Organization and Data Comparison for Model Validation in Faster-than-Real-Time Simulation

Dimosthenis Anagnostopoulos Mara Nikolaidou
Harokopio University of Athens
70 El. Venizelou Str., 17671
Athens, Greece
email: dimosthe@hua.gr *email: mara@di.uoa.gr*

Abstract

Validation is an essential feature of faster-than-real-simulation (FRTS), since models must be validated prior used for near-future predictions. As no relevant approaches exist in the literature, the paper contribution is to propose an approach a) accomplishing validation as a real-time, automated process, with low time overhead, b) dealing with the complexity of validation data, as the number and type of the data compared may be changing and c) realizing the transition from validation process conceptual design to an efficient real-time execution. A formal data organization scheme (data model) and algorithms for constructing and accessing it are implemented for this purpose. Realization of model validation in a FRTS experiment on a single-queue/multi-server processing system is presented to exhibit the applicability of the proposed approach. The proposed organization scheme may also be applied in simulation studies where timing and automation requirements are not critical.

1. Introduction

When simulation reaches conclusions for systems behavior in real time, it is known as *real-time* simulation. In faster-than-real-time simulation (FRTS), results are delivered earlier than real-time. FRTS is thus widely employed for prediction purposes [1]. Performance issues are critical and often addressed using parallel and distributed simulation (PADS) [2]. Communication networks, military networks, transportation systems and manufacturing are application areas where simulation performance in real time is strongly emphasized due to the potential FRTS may offer [2], [3], [4], [5] [6].

A conceptual faster-than-real-time simulation methodology has been introduced in [6], providing a framework for conducting experiments dealing with the complexity and the hard real-time requirements. The following simulation phases have been identified: *modeling, experimentation and remodeling*. During experimentation, both the system and the model evolve concurrently and are put under monitoring. Data depicting their consequent states are obtained in predetermined, real-time intervals of equal length, called *auditing intervals*. A substantial issue is that *time-dynamic systems* may be subjected to changes at any time point, which involve their structure and operation parameters. To deal with them, the model must be subjected to dynamic remodeling, i.e. in real-time, without recompilation.

Validation is the process of determining whether a simulation model is an accurate representation of the system for the particular objectives of the study [7]. In FRTS, we have the unique capability to use system observations and model results to test model validity and also reach reliable predictions for the future system states. 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. A methodological approach for model validation in FRTS bas been presented in [8], aimed at increasing the level of confidence for simulation predictions concerning the time-dynamic system under study. To achieve FRTS validation, system observations and model data must be compared in real time using a computationally efficient process.

As no relevant approaches exist in the literature, the paper contribution is to propose an approach dealing with the following design and implementation issues of model validation:

1. Accomplishing validation as a real-time, automated process, with low time overhead.
2. Determining the nature of validation data and deal with the complexity encountered, as the number and type of the data compared may be constantly changing. We establish a formal data organization scheme (data model) for this objective, based on the relational model.
3. Realize the transition from the validation process conceptual design to the efficient real-time execution.

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

Data structures for system/model data comparison and algorithms for constructing and accessing these structures are thus introduced.

In section 2, we review the essential requirements for accomplishing model validation in FRTS and, in section 3, an organization scheme (data model) is introduced for the data used in the validation process. In section 4, we describe the *auditing tree* structure and propose algorithms and structures for constructing and accessing it, while in section 5 implementation of the data model is analytically described. In section 6, appropriate comparison techniques are discussed for FRTS. Finally, an example involving the construction of the auditing tree, the selection of comparison techniques and the realization of the data organization scheme for a typical single-queue, multi-server processing system is presented in section 7, while conclusions reside in section 8.

2. Model Validation

Numerous model validation types and techniques have been discussed in the literature. In [9], *conceptual model validation*, *computerized model verification* and *operational validity* are identified. In the case of FRTS, only preconstructed models may be used when modifying a composite model in real time, i.e. without terminating the experiment. To accomplishing validation as a real-time, automated process, a methodological approach needs to be practical, realizing the transition from the validation process conceptual design to efficient real-time execution, eve though functionality may be narrowed to eliminate the need for human intervention. The proposed validation approach is a low-level one, being less generic than other methodological approaches for validation and certification, such as the ones proposed by Balci [10] and Birta [11]. However, focusing exclusively on validation using system observations (according to observational specifications, described in [11]), it may be straightforwardly employed for automating validation and also dealing with timing requirements. The following activities are identified for realizing validation in FRTS (also introduced in [8]):

1. Determine the conditions that may lead to model invalidity (thus, to remodeling). These are denoted as *remodeling conditions*.
2. Determine the elements that need to be monitored for performing comparisons between system observations and model results. *Monitoring variables* are used for this purpose. Considering that k monitoring variables, MV_1-MV_k, are used, the respective values of variable i for the system and the model are denoted as $MV_i.r$ and $MV_i.s$.
3. Realize the transition from conditions to the measures under monitoring (i.e. express remodeling conditions in terms of monitoring variables).
4. Determine how each condition contributes to deciding if the model is invalid.
5. Construct the data structure used for maintaining system and model monitoring variable values (such as the *auditing tree* described in section 4).
6. Form and execute the validation algorithm (or *auditing algorithm).*

Activities 1-4 are performed at a specification phase, while activities 5-6 are executed in real-time.

Remodeling conditions need to explicitly converge to a mathematical expression. Each remodeling condition involves one or many comparisons between specific monitoring variables. The number of monitoring variables corresponding to a single condition depends on current system configuration [8]. Discriminating among remodeling conditions is required, according to whether conditions autonomously cause remodeling or not. In the first case, any such condition causes remodeling when fulfilled. We name them *OR* type conditions. In the latter, many conditions of this type need to be fulfilled to cause remodeling – we name them *AND* conditions [8]. As *AND* conditions may not be equally significant, a *weight* factor may be assigned to each of them to determine the significance of each such condition [8].

Then, as remodeling decision involves both *AND* and *OR* type comparisons, a *global scoring algorithm* is required for determining if remodeling must be performed, through accessing all *AND* comparison weights. Scoring models have been used extensively for model validation purposes [12]. Weights (or scores) are determined subjectively when conducting validation and then combined to determine an overall score for the model. The model is considered as valid when this score is higher than a threshold. In our case, we consider a valid model when lower than the threshold, as described in section 4. To compare monitoring variable values, only one comparison technique can be most appropriate for each monitoring variable. The acceptable comparison parameters (or deviation range) must also be determined, according to each specific monitoring variable type. Comparison techniques are further discussed in section 6.

Finally, a generic and consistent relationship between remodeling conditions, monitoring variables and monitoring variable comparisons must be established to be able to determine automatically: a. the variable comparisons that must be made to examine each condition and b. how each comparison may be effectively performed.

3. Validation Data Organization

To execute validation as a real-time, automated process and determine an efficient organization scheme for

validation data, we introduce a data model for remodeling conditions, monitoring variables, monitoring variable values and comparison techniques. All above are considered as discrete entity types. We use the entity-relationship (E-R) model for establishing a data model due to the formalization it offers. Also, for its declarativity, an SQL-based implementation is employed– however, any other implementation method may be applied as well. *Functional dependencies* are examined to derive the appropriate data model. Forming the data model:

1. Remodeling conditions, monitoring variables and comparison techniques are respectively characterized (primary key) by the condition name (*rcname*), variable name (*mvname*) and technique name (*tname*).
2. *RemodelCondition.rcname->RemodelCondition.rctype* Each condition is of a specific type, either AND or OR, no matter how many monitoring variables it involves. Thus, *rctype* obtains a single value from {AND, OR} value list. Type is functionally dependent on the condition name.
3. *RemodelCondition.rcname->RemodelCondition.weight* As a single condition has a specific degree of significance, all variable comparisons corresponding to the same remodeling condition have a common weight. Weight is functionally dependent on the condition name.
4. *MonVariable.mvname->MonVariable.comp_params* There is a specific comparison parameter for each variable, which may be different for variables of the same condition (not dependent on the condition id). Comparison parameter is functionally dependent on the variable name.
5. *MonVarValue.mvname, MonVarValue.timepoint->MonVarValue.modelvalue MonVarValue.mvname, MonVarValue.timepoint->MonVarValue.systemvalue* Monitoring variable values obtained from both the model and the system are determined by the monitoring variable name and the time point. This is due to the fact that, in FRTS, values for a single variable are obtained for more than one time point, as we are trying to reach predictions for at least *p* intervals ahead of real time. Both *modelvalue* and *systemvalue* may be multivalued and thus are functionally dependent on the combination of the variable name and time point.

Concerning the relationships between these entities:

1. A remodeling condition involves one or many monitoring variables; each variable corresponds only to a single condition. The number r_i of monitoring variables corresponding to a single condition *i* can be different whenever auditing is executed, depending on the current system/model configuration
2. A single comparison technique is best suited for each monitoring variable; more than one monitoring variable may use the same comparison technique
3. A monitoring variable value corresponds to a single variable; more than one value may be maintained for a single variable, for different time points

The corresponding E-R data model is depicted in Figure 1.

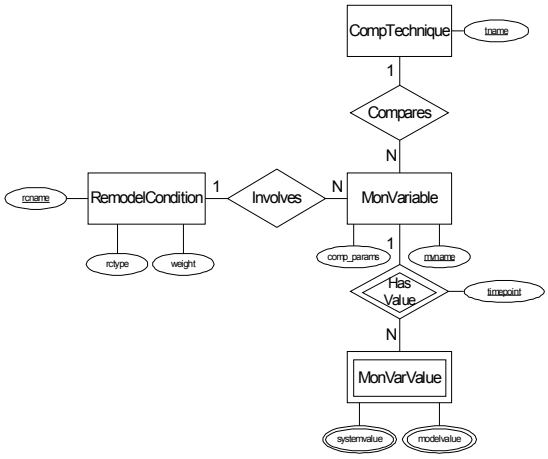

Figure 1: Proposed E-R model for validation data

Transiting from the E-R model to the final relational model, each remodeling condition is ultimately maintained as follows [13], [14]:

`RemodelCondition (rcname, rctype, weight)`

In this relation, *rcname* is unique, *rctype* holds the its type, set to either AND or OR. Note that, technically, attribute *weight* should not be part of this relation, as it is also functionally dependent on *rctype,* as conditions of type OR have a null weight; however, this approach offers simplicity and promotes data manipulation with no overhead. Monitoring variables are maintained as:

`MonVariable (mvname, rcname, tname, comp_params)`

where *mvname* holds the variable name and *rcname* the name of the corresponding condition, *tname* is the respective comparison technique and *comp_params* are the comparison parameters for this specific variable (e.g. the acceptable deviation range, as discussed in section 6). Monitoring variables values are maintained as:

`MonVarValue (mvname, timepoint, systemvalue, modelvalue)`

as *MonVarValue* is a weak entity type [13], [14]. Comparison techniques are maintained as:

`CompTechnique (tname)`

In this data scheme, *MonVariable.rcname* is a foreign key on *RemodelCondition.rcname, MonVariable.tname* on *CompTechnique.tname* and *MonVarValue.mvname* on *MonVariable.mvname*. The relational data model for validation data is overall depicted in Figure 2.

Relation	Attributes
RemodelCondition	rcname, rctype, weight
MonVariable	mvname, rcname, tname, comp_params
MonVarValue	mvname, timepoint, systemvalue, modelvalue
CompTechnique	Tname

Figure 2: Proposed data model for validation data

4. The Auditing Tree

Monitoring variable comparison is realized using the *auditing tree*. It is a conceptual tree structure (that is, it does not follow the formal definition of a tree), divided into two subtrees, which include two corresponding types of end nodes, *OR* and *AND*. Each node corresponds to a single monitoring variable comparison. End nodes of type *OR* represent comparisons that autonomously - if fulfilled - cause remodeling. Nodes of type *AND* are aggregately evaluated to determine if remodeling is required. End nodes are directly accessed from *Root*.

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. End nodes are created and inserted in the appropriate subtree whenever the auditing tree is formed (i.e. during auditing). Thus, the number of tree nodes may be variable, corresponding to the number of comparisons to be accomplished. Each end node is formed for realizing a comparison corresponding to a single remodeling condition. However, a single condition may be expressed via more than one end node. Accessing all nodes, we ensure that all remodeling conditions are evaluated prior to the initiation of remodeling and all reformations/deviations are detected, so that appropriate remodeling actions can be considered. Upon completion of auditing, end nodes are removed.

The auditing tree bears substantial differences from the indicator hierarchy proposed by Balci [10]: It is less generic, expands only to two levels, has two different types of nodes, child nodes have only one parent and the number of nodes differs each time auditing is initiated. Leaf nodes influence directly the validity decision, not indirectly through their parents, and their weight needs not be pre-calculated so that is sums to one per parent.

Key reasons for using this structure are discussed in the following. The auditing tree is oriented towards determining invalidity, which must be examined with minimum time overhead. If some comparisons lead directly to remodeling (type *OR* comparisons), they must be evaluated before all others. Thus, there have to be two node types (*OR*, *AND*). As the number of nodes may be dynamically modified, to avoid reassigning weights in real time, the weight of all comparisons corresponding to the same condition has to be predetermined. As the number of nodes is variable, weights do not thus sum to one per condition. When all comparisons are completed, an aggregate value is calculated and we examine if this exceeds a predetermined threshold. In this way, complexity is transferred from real-time weight assignment to the selection of threshold.

The auditing algorithm is directly derived from the auditing tree. It concludes that the model is invalid if at least one of the a_1 *OR* node comparisons or the aggregate evaluation of the a_2 *AND* node comparisons are fulfilled, that is:

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

Considering there are r conditions causing remodeling (r_o of type OR and r_a of type AND, $r=r_o+r_a$) and k monitoring variables, k comparisons are made. If condition i involves r_i monitoring variables, remodelling decision is based on $k=a_1+a_2$ ($a_1=\sum_{i=1}^{r_o} r_i$, $a_2=\sum_{i=r_o+1}^{r_o+r_a} r_i$) accesses to OR/ AND comparison results, respectively. The auditing tree structure thus enables the consistent realization of the complex comparison process, supporting these essential features:

1. Assigning priorities to specific comparisons (AND, OR nodes) to enable alternative (optimized) algorithms to be effective, i.e. searching the auditing tree for a single condition that may be fulfilled, and then invoke remodeling without accessing the overall tree structure. Such a search would cost considerably less than a_1+a_2.
2. Ensuring that $k=a_1+a_2$ direct accesses to an auditing node structure will be required when all comparison results need to be evaluated (worst-case scenario).
3. Representing each comparison as a separate auditing tree node, the structure of each node includes identification attributes (i.e. condition name and variable name), system and model results, comparison parameters and the weight attribute (only for AND comparisons).

A code fragment for the implementation of the extended node structure as object classes is depicted in Figure 5. As *systemvalue* and *modelvalue* fields may be multivalued, they are represented as arrays of real numbers. Nodes are initialized obtaining the corresponding values of the relational model of Figure 2 and then inserted in the appropriate subtree as direct descendants of *Root*. A code fragment for the implementation of the *Root* node is depicted in Figure 3. Due to the variable number of tree nodes, all nodes (also the root node) are constructed when auditing is initiated

and then are removed.

```
RootNode = OBJECT;
    a1: INTEGER;
    a2: INTEGER;
    ORsubtree: ARRAY INTEGER OF ORnode;
    ANDsubtree: ARRAY INTEGER OF ANDnode;
END OBJECT;
…
VAR
    Root: RootNode;
…
    NEW(Root);
    NEW(Root.ORsubtree, 1..a1);
    NEW(Root.ANDsubtree, 1..a2);
```

Figure 3: *Root* implementation

An algorithm for constructing the auditing tree nodes is depicted in Figure 4. Variable *curr_time* holds the current real time point where auditing is initiated, so that only results concerning this time point are used in auditing.

```
(OR nodes}
create  new_OR_node as
select  RemodelCondition.rcname, mvname, tname,
        comp_params, systemvalue, modelvalue
from    RemodelCondition, MonVariable,
        MonVarValue
where
RemodelCondition.rcname = MonVariable.rcname and
RemodelCondition.rctype = 'OR' and
MonVariable.mvname = MonVarValue.mvname and
MonVarValue.timepoint = $curr_time

{AND nodes}
create  new_AND_node as
select  RemodelCondition.rcname, mvname, tname,
        comp_params, weight, systemvalue,
        modelvalue
from    RemodelCondition, MonVariable,
        MonVarValue
where
RemodelCondition.rcname = MonVariable.rcname and
RemodelCondition.rctype = 'AND' and
MonVariable.mvname = MonVarValue.mvname and
MonVarValue.timepoint = $curr_time
```

Figure 4: Construction of auditing tree nodes

A sample algorithm for implementing auditing is then depicted in Figure 5.

4.1 Null Nodes

A specific case to be considered is the one of null nodes, that is, when either the system value or model value of an auditing tree node is NULL. For instance, as in a GI/G/s system the number of servers (s) may be modified, so is the number of monitoring variables referring to the average service time per server. If a structural modification occurs in the system, *s* will be decreased and the respective model will have an additional server until it adapts to the current condition.

```
FOREACH Node IN ORsubtree
IF Deviates(systemvalue,systemvaluenum,
modelvalue, modelvaluenum,tname,comp_params)
    Remodeling(rcname, mvname);
END IF;
END FOREACH;

FOREACH Node IN ANDsubtree
IF Deviates(systemvalue,systemvaluenum,
modelvalue,modelvaluenum,tname,comp_params)
    CalcWeight (TotalWeight, weight);
    BuildRemodelCondition (RemodelCondition,
rcname, mvname);
END IF;
END FOREACH;

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

Figure 5: Auditing algorithm implementation

At auditing, there will be *s* nodes, one of which will have the following form (if average service time comparison is of type AND):

rcname:	sdelay	*{average service time}*
tname:	single-multiple	*{if 1 system value and n replications}*
mvname:	avg_svcD_s	*{average service time of server s}*
comp_params:	0.3	*{hypothetical value}*
weight:	0.2	*{hypothetical value}*
Systemvalue:	NULL	
modelvalue:	…	*{values from multiple replications}*

Field *modelvalue* would respectively be NULL if the number of servers in the system were increased. As null nodes are caused by structural modifications, they must undoubtedly contribute to remodeling. Thus, *deviates* function must always return true, no matter whether the node type is *AND* or *OR*.

5. Applying the Data Organization Scheme

We discuss how the data organization scheme facilitates the execution of the auditing algorithm. System observations are provided for current interval, while the model produces results for *p* intervals ahead. (Figure 6). As validation is a real-time activity, all preparatory steps must be accomplished prior to its initiation. These steps involve the definition of the remodeling conditions, the comparison techniques and their corresponding attributes. Definitions are forwarded to Auditor tool, which performs the validation process. However, monitoring variables may not be defined at a preparatory stage: their number may be only determined during runtime, as it depends on the current system structure. This is fully supported by the proposed data model.

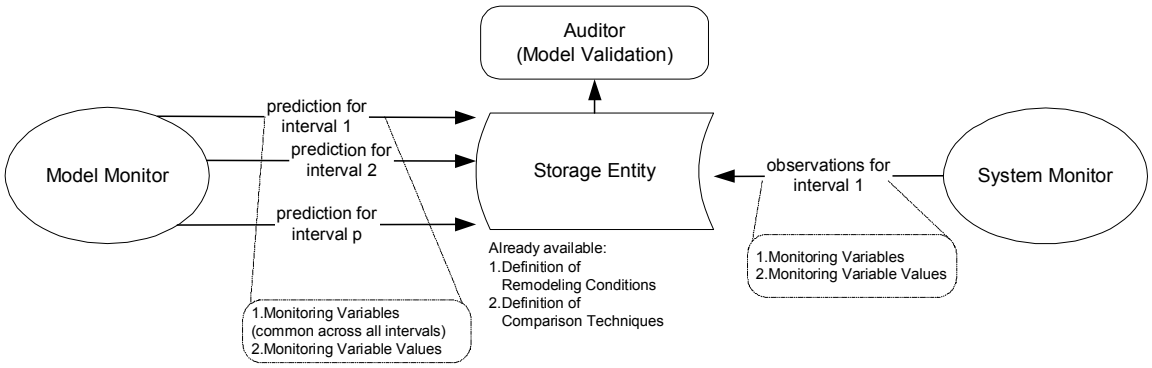

Figure 6: Model results/system observations produced in a single auditing interval

Using the relation database model for validation data, interoperability between different modules is promoted (through common access points) as well as standardization. Applying the proposed organization scheme thus involves the following steps (marked as RT when executed in real time):

Actor	Step	Sample implementation
User (during set up)	1. Set up validation data	
	1.1 Define remodeling conditions (name, type and weight)	`insert into RemodelCondition values ($rcname, $rctype, $weight)`
	1.2 Define comparison techniques (name)	`insert into CompTechniques values ($tname)`
	1.3 Implement a respective comparison function for each technique defined	
Model Monitor	2. Validation	
	2.1 Define monitoring variables (RT) (name, comp. parameters, rem. condition name and technique name)	`insert into MonVariable values ($mvname,$comp_params,$rcname, $tname)`

	2.2 Store model values for all monitoring variables and all predicted time points (RT)	`foreach pred_timepoint`
		`  foreach monvariable`
		`  insert into MonVarValue`
		`  values ($mvname, NULL,`
		`  $modelvalue, $pred_timepoint)`
		`  end foreach`
		`end foreach`

System Monitor	3. Validation	

	3.1 Store system values for all monitoring variables (RT)	`foreach MonVariable`
		`  select count(*) into $found`
		`  from MonVarValue`
		`  where timepoint = $curr_time`
		`  and mvname = $mvname`
		`  if $found = 1`
		`  update MonVarValue`
		`  set systemvalue = $systemvalue`
		`  where mvname = $mvname`
		`  and timepoint = $curr_time`
		`  else`
		`  insert into MonVarValue`
		`  values ($mvname, $systemvalue,`
		`  NULL, $curr_time)`
		`end foreach`

In an SQL like notation, *$var* gives the value of variable *var* and *curr_time* returns the current real-time point. SQL does not support the ARRAY data type used for storing multiple values of *modelvalue* and

systemvalue. However, this functionality may easily provided using the STRING data type and converting values to/from strings.

Following the above steps, model validation involves constructing the auditing tree nodes (Figure 7), and accessing the tree to apply the auditing algorithm (Figure 8), both being executed in real time.

6. Comparison Techniques

We propose the following three techniques as most appropriate for realizing monitoring variable comparison:

1. System - model value comparison, for single-valued variables (i.e. when only one value is available from the model and the system).
2. Inspection approach, for statistical variables when available one system observation data set and *n* model result data sets [15]. In FRTS, it is evident that only a single system data set will be available in almost all cases, as system observations are produced within a single auditing interval.
3. Confidence interval approach, for statistical variables when available *m* system observation data sets and *n* model data sets) [15]. We suggest the classical approach proposed by Welch for building a confidence interval based on a different number of independent data sets [16], as other approaches are more restrictive, such as the paired-t approach [15], imposing that $n=m$, which can be only rarely ensured.

For single-valued variables, system and model variables ($MV_i.s$, $MV_i.r$) are directly obtained. For statistical variables with a single system observation data set and *n* model data sets,

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

where $MV_{ij}.s$ is the statistical sample obtained from replication *j* when *n* replications are made (i.e. in the case of terminating simulations).

In the third case, where statistical variables with *m* system observation data sets and *n* model data sets are available, we build a confidence interval based on a different number of independent data sets [16]. According to the Welch approach:

$$\overline{MV_i.r} = \frac{\sum_{j=1}^{m} MV_{ij}.r}{m},\quad \overline{MV_i.s} = \frac{\sum_{j=1}^{n} MV_{ij}.s}{n}$$

$$S^2(MV_i.r) = \frac{\sum_{j=1}^{m} [MV_{ij}.r - \overline{MV_i.r}]^2}{m-1}$$

$$S^2(MV_i.s) = \frac{\sum_{j=1}^{n} [MV_{ij}.s - \overline{MV_i.s}]^2}{n-1}$$

The estimated degrees of freedom are computed as

$$f = \frac{[S^2(MV_i.r)/m + S^2(MV_i.s)/n]^2}{[S^2(MV_i.r)/m]^2/(m-1) + [S^2(MV_i.s)/n]^2/(n-1)}$$

The following interval as an approximate *100(1-a)%* confidence interval for $MV_i.r$-$MV_i.s$

$$\overline{MV_i.r} - \overline{MV_i.s} \pm t_{f,1-a/2}\sqrt{\frac{S^2(MV_i.r)}{m} + \frac{S^2(MV_i.s)}{n}}$$

Evidently, the deviation range defines the value *a*, meaning that we wish the confidence interval to cover $MV_i.r$-$MV_i.s$ with probability *1-a*. Suppose that the upper and lower endpoints of the interval are marked as *u(a)* and *l(a),* respectively. If $0 \notin [l(a), u(a)]$, the difference between $MV_i.r$ and $MV_i.s$ is statistically significant at level *a* and we consider the model to be invalid.

7. A Multi-Server Processing System Case Study

A single-queue, multi-server processing system is used as an example involving validation, data organization and model-system observation comparison offering the capability to apply all three comparison techniques. This system is modeled as a *GI/G/s* queue, according to classic queuing theory, where GI (general independent) is the distribution of interarrival times, G (general) is the distribution of service times and *s* is the number of servers (we consider that s>1). The system has a variable number of servers that can be modified during runtime, as servers may be abruptly activated or de-activated. Servers have identical service characteristics. The objective of FRTS is to reach reliable conclusions and to ensure model validity taking into consideration system changes. We consider that *n* model replications are executed for reaching reliable simulation results, i.e. the case of terminating simulations, where the model does not reach a steady state. Two remodeling conditions are defined:

(1) Different number of servers (*diffserv*) as of type OR due to its significance.

There is as single monitoring variable *server_no* for condition *diffserv*. The model value for this variable is single-valued, as all replications are performed with the same number of servers. The allowed deviation range (*comp_params*) is set to 0.0, i.e. between the number of servers in the system and the model. As *server_no* is single-valued, the *single* comparison technique may be employed.

(2) Deviation in the average service time per server (*sdelay*) as of type AND (having a 0.2 weight)

There are *s* monitoring variables for condition *sdelay* - each corresponding to the delay of each one of the *s* servers. Deviation range (*comp_params*) is set to 0.1 for all such comparisons. As one observation data set

and *n* model data sets (stemming from the *n* replications) are available, a *single-multiple* comparison must be performed.

Organizing validation data, relations *RemodelCondition, CompTechnique* and *MonVariable* are formed as described in Table 1, Table 2 and Table 3, respectively. The following steps are automated.

Table 1: Relation *RemodelCondition*

rcname	rctype	weight
diffserv	OR	-
sdelay	AND	0.2

Table 2: Relation *CompTechnique*

tname
single
single_multiple
confidence_interval

Table 3: Relation *MonVariable*

mvname	rcname	tname	comp_params
server_no	diffserv	single	0.0
avg_svcD$_1$	sdelay	single_multiple	0.1
avg_svcD$_2$	sdelay	single_multiple	0.1
…			
avg_svcD$_s$	sdelay	single_multiple	0.1

Examining condition (1), the model is considered as valid when:

$$server\_no.s \in [server\_no.r\ (1\text{-}0.0),\ server\_no.r(1+0.0)] \Leftrightarrow server\_no.s = server\_no.r$$

Examining condition (2) for monitoring variable *i*, the model is considered as valid when:

$$avg\_svcD_i.s \in [avg\_svcD_i.r(1\text{-}0.1), avg\_svcD_i.r)(1+0.1)] \Leftrightarrow avg\_svcD_i.s \in [0.9*avg\_svcD_i.r,\ 1.1*avg\_svcD_i.r)]$$

where $avg\_svcD_i.s = sum(avg\_svcD_{i1}.s,\ avg\_svcD_{i2}.s,\ \ldots,\ avg\_svcD_{in}.s)/n$ and $avg\_svcD_{ij}.s$ is the average delay of server *i* in replication *j*. As two conditions are examined, there are two types of auditing tree nodes for comparing model results and system observations (Table 4).

The actual number of nodes depends on the current system configuration. According to the remodeling conditions, there will be one node of type OR and *s* nodes of type AND. However, *s* can be modified when the system is subjected to structural changes.

We executed a number of experiments with a M/M/5 system (exponential interarrival times, exponential service time, *s=5*) with *n=10* (10 replications are made). The execution platform is a Sun Ultra 5 with 1 CPU and 640Mbyte running Solaris 8. Modsim III is used for model implementation. For interarrival times, $\lambda=0.5$; for service times, $\lambda=4.0$. Threshold (Figure 5) was set to 0.5. An auditing tree instance is depicted in Figure 7. The corresponding system-model data comparison results are presented in Table 5.

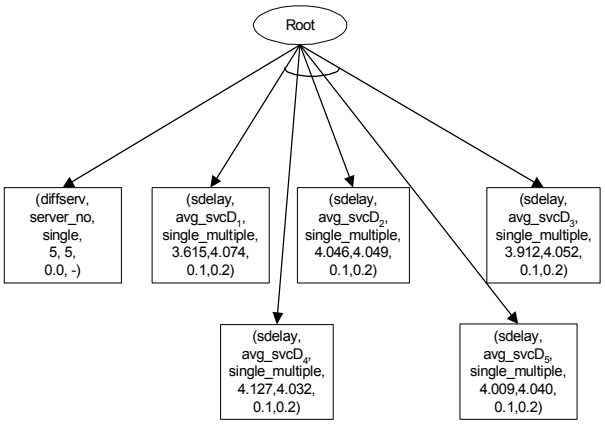

Figure 7: Auditing tree instance

Table 4: Auditing tree node types

node type		rcname	mvname	system value	model value	tname	comp_ params	weight
1	OR	diffserv	server_no	single value {system observation}	single value {used in all replications}	single	0.0	-
2	AND	sdelay	avg_svcD$_i$	single value {from system observations}	multiple values {replication results}	single_multiple	0.1	0.2

Table 5: System-model data comparison

mvname	rcname	tname	model value	system value	lower endpoint	upper endpoint	valid
server_no	diffserv	single	5	5	5.0	5.0	yes
avg_svcD$_1$	sdelay	single_multiple	4.074	3.615	3.254	3.977	no
avg_svcD$_2$	sdelay	single_multiple	4.049	4.046	3.641	4.451	yes
avg_svcD$_3$	sdelay	single_multiple	4.052	3.912	3.521	4.304	yes
avg_svcD$_4$	sdelay	single_multiple	4.032	4.127	3.714	4.540	yes
avg_svcD$_5$	sdelay	single_multiple	4.040	4.009	3.608	4.410	yes

Concerning *diffserv*, the number of servers is not changed and the OR condition is not fulfilled. Concerning *sdelay*, model invalidity is only indicated in the case of the average delay of server1 (avg_svcD_1). Thus *TotalWeight = 0.2*. A threshold equal to 0.5 requires at least two invalid AND nodes to cause remodeling.

To apply the third comparison technique, we consider a composite system consisting of *m* identical *GI/G/s* components, such as the ones discussed above. All subsystems have identical interarrival and service characteristics, thus it is possible to consider that *m* independent observation sets are available from the system. In this case, we follow the approach for constructing an approximate *100(1-a)* percent confidence interval for $avg\_svcD_i.r - avg\_svcD_i.s$, for each monitoring variable *i*, having previously calculated $\overline{avg\_svcD_i.r}, \overline{avg\_svcD_i.s}, S^2(avg\_svcD_i.r), S^2(avg\_svcD_i.s)$ and f. For example, in a system consisting of *m=5* M/M/5 components, there are 5 values for avg_svcD_1 (i.e. server1 of each of the 5 components). Considering *m=5* system values (3.90, 4.02, 3.97, 3.89, 4.15) and experimental results from *n=10* replications (3.92, 4.05, 3.91, 4.08, 4.24, 4.08, 4.17, 4.01, 4.14, 3.99) for avg_svcD_1, the 90% confidence interval constructed for $avg\_svcD_1.r\text{-}avg\_svcD_1.s$ is *[-0.179, 0.027]*. As $0 \in [l(a), u(a)]$, the difference between the two means is not statistically significant at level *10%*, and the model is determined to be valid.

8. Conclusions

The advantages of the proposed data organization scheme may be summarized as follows:

1. Automated execution of model validation activities
2. Distinguishing the most significant conditions that cause remodeling and using a different comparison technique, depending on the specific data sets under comparison
3. Standardization of the way system and model data are maintained and processed
4. Ensuring a low time overhead for executing auditing, especially in the case of a large amount of multiple-valued monitoring variables

The last point refers to application domains where multiple (e.g. thousands) monitoring variables are used to compare the corresponding system and modes states. Such domains are computer networks, where sessions are initiated, transfer data and then terminated. Despite its suitability for FRTS, the proposed organization scheme may as well be applied for consistently performing validation in simulation studies where timing and automation requirements are not so critical.

References

[1] Carothers C., "XSim: Real-Time Analytic Parallel Simulations", in *Proceedings of PADS'02, IEEE Computer Press*, 2002

[2] Fujimoto R., D. Lunceford, E. Page, A. Uhrmacher, "Grand Challenges for Modeling and Simulation", Dagstuhl Report, 2002, http://www.dagstuhl.de/About/index.en.html

[3] Perumalla K., R. Fujimoto, T. McLean, G. Riley, "Experiences Applying Parallel and Interoperable Network Simulation Techniques in On-Line Simulations of Military Networks", in *Proceedings of PADS'02, IEEE Computer Press*, 2002

[4] Hu K., M.Takai, J. Martin, Bargodia R., "Looking Ahead of Real Time in Hybrid Component Networks", in *Proceedings of PADS'01, IEEE Computer Press*, 2001

[5] Schreckenberg M., L. Neubert, J. Wahle, "Traffic Simulation: Simulation of Traffic in Large Road Networks", *Future Generation Computer Systems*, vol. 17, 2001, pp. 649–657

[6] 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

[7] Balci O., "Verification, Validation and Accreditation o Simulation Models", in *Proceedings of WSC'97*, *IEEE Computer Press*, 1997

[8] Anagnostopoulos D., "A Methodological Approach for Model Validation in Faster-than-Real-Time Simulation ", *Simulation Practice and Theory*, Elsevier Science, vol. 10, no. 3-4, 2002, pp. 121-139

[9] Sargent R., "Verification, Validation and Accreditation of Simulation Models", in *Proceedings of WSC'00*, *IEEE Computer Press*, 2000

[10] Balci O., "A Methodology for Certification of Modeling and Simulation Applications", *ACM TOMACS*, vol. 11, no. 4, 2001

[11] Birta L., F. Ozmizrak, "A Knowledge-Based Approach for the Validation of Simulation Models: The Foundation", *ACM TOMACS*, vol. 6, no. 1, 1996

[12] Balci O., "How to Assess the Acceptability and Credibility of Simulation Results" in *Proceedings of WSC'89*, *IEEE Computer Press*, 1989

[13] Date C., *An Introduction to Database Systems*, Pearson Addison Wesley, 2003

[14] Elmasri R., S. Navathe, *Fundamentals of Database Systems*, Addison-Wesley, 1992

[15] Law A.M., W.D. Kelton, *Simulation Modeling and Analysis,* McGraw Hill, 2000

[16] Welch B. L., "The Significance of the Difference Between Two Means when the Population Variances are Unequal", *Biometrica*, vol. 25, 1938, pp. 350-36