

ASSESSMENT OF SIMULATION MODELS BASED ON TRACE-FILE ANALYSIS: A METAMODELING APPROACH

Juri Tolujev

Faculty of Automation
and Computing Techniques
Dept of Modeling and Simulation
Riga Technical University
LV-1658 Riga, LATVIA

Peter Lorenz
Daniel Beier

Faculty of Computer Science
Institute for Simulation and
Graphics
University of Magdeburg
D-39106 Magdeburg, GERMANY

Thomas J. Schriber

Michigan Business School
Computer and
Information Systems
The University of Michigan
Ann Arbor, MI 48109, U.S.A

ABSTRACT

Many important characteristics of simulation models, including queuing models, can be investigated by the use of metamodels. Problems in qualitative analysis such as analyzing model dynamics and coming to a careful understanding of model behavior can be dealt with this way. Metamodels can provide precise results even for quantitative analysis tasks, such as those involving the movement of dynamic model elements. This paper describes the use of a type of metamodeling to support the assessment of simulation models based on the analysis of trace files produced at the time of model execution. Because of the simple structure of these trace files, a simulation model can create them easily. The analysis and interpretation of trace files that is described here is independent of the simulation language used to create the original model.

The tools presented in this article can be used for these purposes:

- to construct generic model structures at the metamodel level and then animate aspects of model behavior in terms of these structures;
- to build a graphic display indicating which dynamic model elements moved at which times between which points in the model, and in which real-time order in cases of time ties;
- to determine when (and if) user-specified model conditions come about; and
- to develop statistical information that might not have been planned for in the design of the original model.

Future plans call for making these tools available in a World Wide Web environment to support assessment of simulation models.

1 INTRODUCTION

Most commercial simulation systems offer a limited set of tools with which to analyze the structure and behavior of simulation models (Banks 1996). The tools and methods provided apply only to relatively simple, well-known classes of dynamic processes, however. For periodic or other nonstationary processes, other tools and methods are needed. Simulation software typically doesn't support automatic detection and reporting of these more complex types of processes. Inexperienced modelers can easily overlook the presence of such processes in their models, and might therefore fail to analyze the behavior of their models correctly. A modeler needs special analytic tools that support detailed examination of dynamic processes in such cases, as well as in more routine cases, to come to a better understanding of model behavior.

For example, consider subtle situations such as those described in Schriber and Brunner (1996; 1998), in which event sequences depend on the design of the original modeling software (e.g., SIMAN vs. ProModel vs. GPSS/H) and cannot be easily predicted unless the modeler is an expert in the software being used. Such situations can be analyzed in language-independent fashion with use of the tools presented in this paper.

More generally, the tools presented here can support model assessment on the part of an *independent modeling expert*. Such an expert can play an important role in verifying and validating simulation models (Arthur and Nance 1996). Techniques suggested in the literature for model verification are numerous (Sargent 1996) and for non-experts can be daunting. For example, one categorization of such techniques leads to 15 principles and 45 methods (Balci 1995). Other authors have also delved into the subject (e.g., Law and Kelton 1991). When assessing a model, a modeling expert has to justify the choice of verification and validation methods and

demonstrate correct implementation of the methods. The expert must identify and understand the type of process being modeled and must be aware of any special aspects of the process as well.

Frantz (1995) has suggested seventeen techniques for assessment of simulation models. Among these, only the so-called *metamodeling* technique is based on experimental inspection of the original model. In Barton (1994) and Caughlin (1997) the term *metamodel* is used to designate an algebraic model that relates output values to a simulation model's input factors. Huber (1996) has extended classes of metamodels to include those based on fuzzy graphs and neural networks. Models based on trace-file data are members of a new class of metamodels called *dynamic metamodels*. Such algorithmic and executable models are able to reconstruct an original model's behavior through analysis of trace-file data (Tolujev 1997b).

The new class of metamodels introduced here is characterized by metamodels that are created automatically and empirically and are based on trace-file analysis. The class is of a kind that addresses the dynamic aspects of simulations performed with the original model. The methodology creates and then exercises the metamodel. Methods for complete identification of metamodels in this class are yet to be developed.

2 THE STRUCTURE OF METAMODELS DESIGNED FOR TRACE-FILE ANALYSIS

Queuing systems are frequently represented as taking the form of interactions among *stations* (static elements forming the system layout) and *transactions* (the dynamic elements that move from station to station). If this worldview is adopted, information about events that change the position of transactions in a model is sufficient to support analysis of queuing system characteristics.

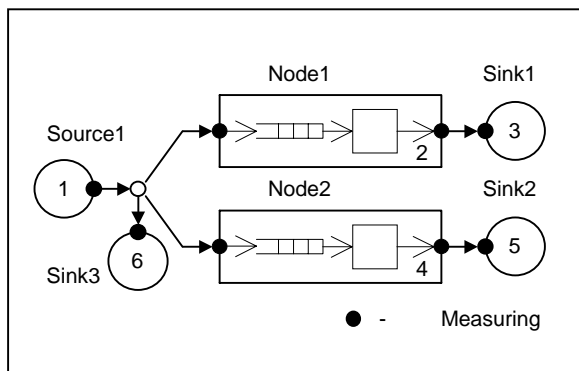


Figure 1: Display of a Metamodel Structure of a Typical Queuing System

Only a small set of station classes and a description of their interconnections are needed to determine the structure of

the type of metamodel discussed here. The set of station classes used to represent such metamodels consists of *sources*, *sinks* and *nodes*, as depicted for a specific case in Figure 1.

An object of the class *node* models transaction delays. Nodes represent elements of various complexities: serial or parallel channels, queues, storage points, and so on. *Nodes* and *sinks* might have any number of input channels. *Nodes* and *sources*, however, have only one exit. Sources and sinks do not delay transactions during their creation or destruction. If such delays take place, they are modeled with nodes.

The processing of a trace file requires that the file consist of records composed of these four fields: Field 1 (time), Field 2 (transaction ID), Field 3 (station ID), and Field 4 (event type, e.g., input/output). A corresponding record must be created and written into the trace file each time a transaction reaches a measuring point when an instrumented version of the original model is being executed. The instrumentation of the original model (that is, the insertion of measuring points into it) can be accomplished automatically by software, as described in Section 3.

The instant in simulated time at which a record is created and written into the trace file is denoted as $t_{in/out}^i(Tr)$. Fields 1, 2, 3 and 4, as described above, have the values t , Tr , i and in/out , respectively. If $t_{out}^i(Tr) \neq t_{in}^j(Tr)$ for two neighboring components i and j , then so-called *connecting nodes* are inserted into the metamodel.

The records in the trace file and possibilities for identifying structures depend on the positioning of measuring points in the original model. Therefore all input and output channels are equipped with measuring points. The identification step results in complete reconstruction of the original model's structure. It is obvious that only components that are represented by events in the trace file can be taken into account in the reconstruction process. For example, node 2 in Figure 1 does not come into play and so might be completely ignored if the input stream is not very intense and if transactions go to node 2 only when and if 10 transactions are located at node 1.

3 TOOLS FOR ANALYZING MODELS OF QUEUING SYSTEMS

The concepts sketched briefly in Section 2 have been made operational through development of a multi-component tool set. This tool set consists of a Model Editor, a Trace Editor, a Proof Generator, a Trace Parser, and a Trace Viewer, as shown in Figure 2. With the exception of the Model Editor, each of these components is independent of the simulation software used to develop the original simulation model (the model whose characteristics are being analyzed). This cannot be true of the Model Editor itself, however, for reasons explained below. As suggested

in Figure 2, the Model Editor on which work to date has been based is specific to the use of GPSS/H™ (Crain 1997) as the modeling language.

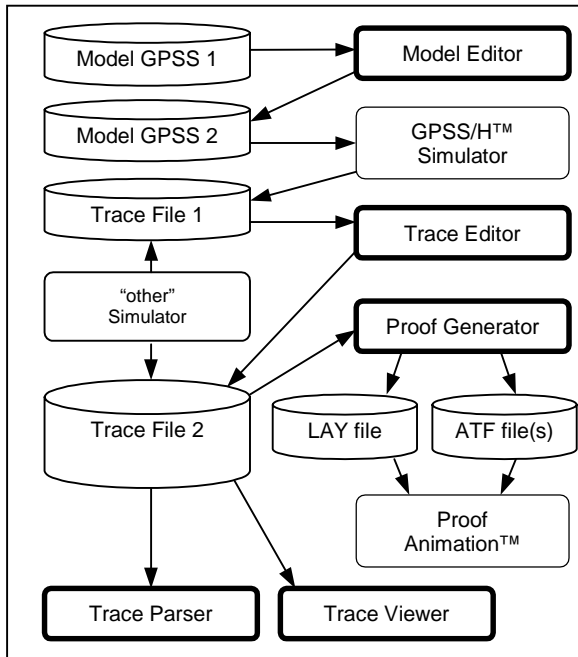


Figure 2: The Roles Played by the Model Editor, Trace Editor, Proof Generator, Trace Parser, and Trace Viewer in Model Assessment Via Trace-File Analysis

3.1 Model Editor

The role played by the Model Editor is to read the model whose characteristics are to be analyzed, and then create a variation of this model that has been instrumented with the measuring points needed to create a trace file. This role is shown at the top of Figure 2, where “Model GPSS 1” is the original model, and “Model GPSS 2” is the instrumented variation of it. A Model Editor must be able to deal with the syntax and semantics of the language used to create the original model, and so cannot be language independent.

The instrumented model is created by the Model Editor using the syntax of the original modeling language. Another simulation is then performed with the instrumented version of the model, producing a trace file (“Trace File 1” in Figure 2).

Some of the particulars of building the Model Editor specific to GPSS/H models are discussed in Section 4.

3.2 Trace Editor

The Trace Editor of Figure 2 reads the trace file produced during the simulation (“Trace File 1”) and produces a reformatted version of it (“Trace File 2”). The reformatted

version supports follow-on analysis performed by the Proof Generator, whose role is discussed below.

The Trace Editor provides the possibility of choosing among three levels of resolution in the metamodel:

- High Resolution Metamodel (encompasses the details of all possible model elements that are distinguishable in a trace file)
- Middle Resolution Metamodel (encompasses sources and sinks, and complete paths and loops)
- Low Resolution Metamodel (encompasses sources and sinks, but otherwise represents the remaining parts of the original model as a single element)

3.3 Proof Generator

The Proof Generator of Figure 2 reads Trace File 2 and creates a metamodel structure in a canonical form as a basis for showing the animated movement of transactions from node to node. This structure is stored in two types of files: a LAY (“layout”) file; and one or more ATF (“animation trace file”) files. These files, in turn, are inputs to Proof Animation™ (Henriksen 1997), which is commercial animation software used to provide an animation of the metamodel. A snapshot taken from such an animation is shown in Figure 3.

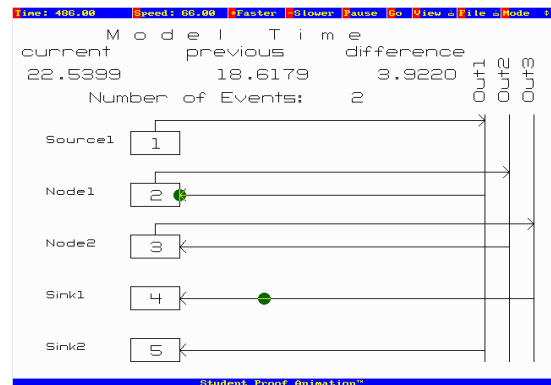


Figure 3: Snapshot of an animation produced by use of the metamodel

The animation can be viewed in either of two modes. Mode A shows direct representation of the event order as stored in the trace file. Transactions change their positions in steps at distinct points in simulated time, but possibly also at identical times (when time-ties are involved).

In alternative Mode B, simulation time is shown on the screen in terms of model time. Only one transaction is moved at a time, even if two or more transactions move at the same simulated time. Transactions move discretely and

continuously to provide the user with an understanding of the paths along which the movement is taking place

3.4 Trace Viewer

The Trace Viewer of Figure 2 inputs Trace File 2 and produces a three-axis graphical representation of queuing-system process dynamics, as shown in Figure 4. The movement of transactions from node to node is shown relative to time in the “transfers dimension.” The “transactions/node dimension” displays the transactions that captured or are waiting at a node. Both windows are modified synchronously because the time axes are equally scaled. These time axes are shown in descending order which clarifies the connection between both dimensions.

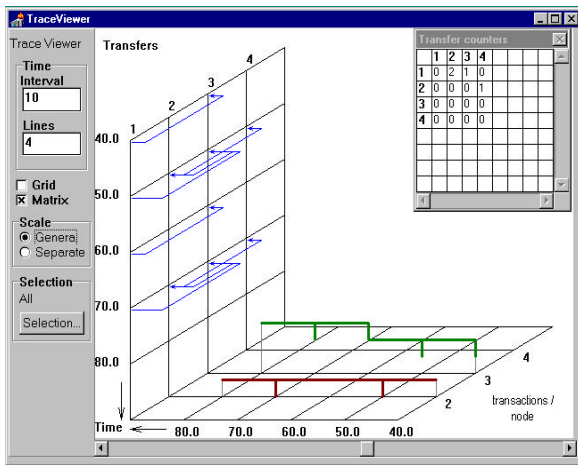


Figure 4: An Example of the Representation of Model Dynamics Produced by the Trace Viewer

By changing the time scale, one can choose between the representation of single events or entire panoramas (level of detail). The *time interval* axis offers the possibility of changing the density of displayed events. The possibilities for displaying and seeing a larger number of elements simultaneously are limited, because the metamodels consist of fewer elements than the original models from which they are extracted. Any subset of model components can be displayed for analysis via a corresponding selection of component numbers. Individual transactions can be selected to show their passage through the model. The “transfer counts” (given in the matrix at the top of Figure 4 show the count of transitions between model components, updated as of the displayed time.

3.5 Trace Parser

The Trace Parser of Figure 2 uses Trace File 2 as a “database” and conducts a statistical analysis of simulation

data. Advanced search features are provided to help identify statistical phenomena that are out of the ordinary.

Three types of statistics are produced by the Trace Parser in standard format:

- (1) The data is collected and calculated for all metamodel elements and displayed as Component Statistics. The display shows the following:
 - number of incoming and outgoing transactions;
 - current, average and maximum node contents; and
 - the distribution of transaction delay times.
- (2) Inter-Arrival Statistics are computed for each connection between elements, including
 - number of transfers;
 - time of first and last transfer; and
 - the distribution of the inter-transfer times.
- (3) The stream of transactions originating at a source is analyzed, source by source, and the results are displayed as Transaction Statistics. These statistics describe:
 - element chains as complete paths (from source to sink) or loops in the metamodel; and
 - completion-time distributions for paths and loops.

The following examples are suggestive of the type of information that the Trace Parser can extract from Trace File 2:

- Type 1. Find the simulated time or times when:
- a transaction leaves component *a*;
 - the transaction count in node *a* equals *m*;
 - a transaction enters node *a* and node *b* is unused.

- Type 2. Find the simulated time intervals when:
- node *a* is unused;
 - the transaction count in node *a* equals *m*;
 - node *a* is used and node *b* is unused.

- Type 3. Find the following user-specified output data:
- the number of transactions processed by *a*;
 - percent of the time that node *a* was in use;
 - distribution of transaction delay time at node *a*.

The search function can be applied globally across the entire simulation, or it can be applied locally to a specified interval of simulated time. It is possible for a Type 3 search to display the results in the form of time lines and corresponding line diagrams. It is also possible to inspect logically complicated situations in single-step mode and in the form of an animation.

4 DESIGN OF THE MODEL EDITOR FOR GPSS/H

Some of the details of the design of the Model Editor specific to GPSS/H will now be sketched. As shown in Figure 2, the Model Editor automatically instruments a GPSS/H model, generating new source code that will carry out the simulation as originally specified, and that will produce Trace File 1 of Figure 2 as well. The Model Editor inserts measuring points in the original model after source-code analysis. These points are located at the connections between model components. An additional standard “trace selector” (expressed in GPSS/H source code) is appended to record the relevant data.

This automatic GPSS/H model modification is based on the following considerations used in design of the Model Editor:

- Sources and sinks correspond to the GPSS/H GENERATE, TERMINATE, SPLIT and ASSEMBLE blocks.
- Nodes are determined by identifying:
 - GPSS elements used to model equipment (Facilities and Storages);
 - other potential points of delay for transactions (e.g., refusal-mode TEST and GATE blocks).

The list of all GPSS/H block statements modified by the Model Editor and the corresponding format used for the modification are shown in Table 1. Note that two different formats are needed for the modification of ADVANCE blocks because each transaction passes the trace selector before and after a time delay at an ADVANCE block.

The Model Editor extends the data structure of GPSS/H transactions by adding transaction parameters named BLOCKTYP, KOMPID, ASMCOPY, AADR1, and AADR2. The subroutine “trace selector” is accessed by the block names ATRA1, ATRA3 and BLOASM. The identification of each GPSS/H equipment-modeling component, as determined from format rules 3, 4 and 5 in Table 2, is stored in the transaction parameter KOMPID. The term *b* is used for the whole operand section of GPSS/H blocks beginning with the B Operand.

Table 1: GPSS/H Block Statements, With Their Block-Type (BT) Codes and Modification Format Numbers

GPSS/H Block Statement	BT Code	Modification Format No.
SEIZE	11	3 or 4
PREEMPT	21	3 or 4
ENTER	31	3 or 4
QUEUE	41	3 or 4
RELEASE	12	3 or 4
RETURN	22	3 or 4
LEAVE	32	3 or 4
DEPART	42	3 or 4
LINK	51	5
GENERATE	64	1
TERMINATE	71	2
pre ADVANCE	83	2
post ADVANCE	82	1
TEST, GATE, GATHER, MATCH	93	2
TRANSFER ALL, TRANSFER BOTH	103	2
SPLIT	114	6
ASSEMBLE	121	7

The BT Code of Table 1 is the code for block types used in some of the modification formats. The modification formats themselves are given in Table 2, where column 1 (MF No.) repeats the Modification Format No. of Table 1. (Please contact the authors for further details.)

Table 2: The Model Editor Modification Formats

MF No.	GPSS/H Text Before Modifications	GPSS/H Text After Modifications
1	<i>BLOCKNAME a,b</i>	<i>BLOCKNAME a,b</i> ASSIGN BLOCKTYP, <i>BTcode</i> ,PH TRANSFER SBR,ATRA1,(AADR1)PH
2	<i>BLOCKNAME a,b</i>	ASSIGN BLOCKTYP, <i>BTcode</i> ,PH TRANSFER SBR,ATRA1,(AADR1)PH <i>BLOCKNAME a,b</i>
3	<i>BLOCKNAME a,b</i> <i>a</i> is a standard symbol or expression, computing the value of PH(KOMPID).	ASSIGN BLOCKTYP, <i>BTcode</i> ,PH ASSIGN KOMPID, <i>a</i> ,PH <i>BLOCKNAME PH(KOMPID),b</i> TRANSFER SBR,ATRA1,(AADR1)PH
4	<i>BLOCKNAME a,b</i> <i>a</i> is a standard symbol, assigning a constant value to PH(KOMPID).	<i>BLOCKNAME a,b</i> ASSIGN KOMPID, <i>a</i> ,PH ASSIGN BLOCKTYP, <i>BTcode</i> ,PH TRANSFER SBR,ATRA1,(AADR1)PH
5	LINK <i>a,b</i>	ASSIGN BLOCKTYP, <i>BTcode</i> ,PH ASSIGN KOMPID, <i>a</i> ,PH TRANSFER SBR,ATRA1,(AADR1)PH LINK <i>a,b</i>
6	SPLIT <i>a,b</i>	TRANSFER SBR,ATRA3,(AADR1)PH SPLIT <i>a,b</i>
7	ASSEMBLE <i>a</i>	ASSIGN ASMCOPY, <i>a</i> ,PH TRANSFER SBR,BLOASM,(AADR2)PH ASSEMBLE <i>a</i>

5 AN EXAMPLE OF MODEL ASSESSMENT

Even very simple GPSS/H (and other) models of queuing systems might contain “secrets” that are difficult to explore based on direct analysis of the model itself. Consider Figure 5, which shows a routine GPSS/H model of a one-line, one-server system as a beginner might model it. Clients arrive at the service point and, if the server is in a state of capture, they go to the back of a user chain (a list composed of clients waiting their turn for service). When the server finishes the ongoing service, the next waiting client is removed from the user chain with the intention that it is to capture the server.

```

GENERATE 10
GATE FU JOE,BJOE
LINK CLIENTS,FIFO
BJOE SEIZE JOE
ADVANCE 20
RELEASE JOE
UNLINK CLIENTS,BJOE,1
TERMINATE 1

```

Figure 5: Model of a One-Line, One-Server System

The builder of the deterministic Figure 5 model might think that the model implements a first-come, first-served service order. But does it? Are there times in this model when the service order is other than first-come, first-served? Yes, there are such times (simulated time 30 is such a time, as we show below), but it is not easy even for an experienced user of GPSS/H to reach this conclusion by direct inspection of the model itself, and without knowledge of the underlying algorithms followed by GPSS/H. The conclusion is easily reached, however, by model assessment through trace-file analysis, as will now be demonstrated.

The methodology outlined in Figure 2 was used to process the model of Figure 5, producing a metamodel composed of these numbered elements:

- 1 - Source GENERATE
- 2 - Facility JOE
- 3 - User chain CLIENTS
- 4 - Sink TERMINATE

The Trace Viewer was then used to produce the Figure 6 display of simulated events taking place early in the simulation. Simulated time is shown on the vertical axis in Figure 6, and model-element numbers are shown on the “horizontal” axis. (Model-element number 1 corresponds to “Source GENERATE” as listed above, for example.) Element-to-element transfers are represented in Figure 6 with lines that protrude from the “time-element” plane, span the distance between the two elements involved, and

then go back into the “time-element” plane in the form of an arrowhead. For example, we see in Figure 6 that at simulated time 10.0 (“10.0” is not shown on the time axis, to avoid clutter), there is a transfer from element 1 to element 2. (This transfer takes place when a unit of traffic enters the model at time 10.0 and captures the server without delay.) We also see in Figure 6 that at simulated time 20.0, there is a transfer from element 1 to element 3. (This transfer takes place when a unit of traffic enters the model at time 20.0 and goes onto the user chain to wait its turn to use the server.)

When there are two or more transfers at a given simulated time, the real-time order of the transfers is represented in terms of how far the transfer line protrudes from the “time-element” plane. The further out a transfer line protrudes, the later in real time the transfer occurs. At time 30.0 in Figure 6, for example, we see that two transfers take place: a transfer from element 1 to element 2, and a transfer from element 2 to element 4. The transfer from element 2 to element 4 takes place first, then the transfer from element 1 to element 2 takes place. (The transfer line from element 2 to element 4 does not protrude as far from the “time-element” plane as the transfer line from element 1 to element 2.)

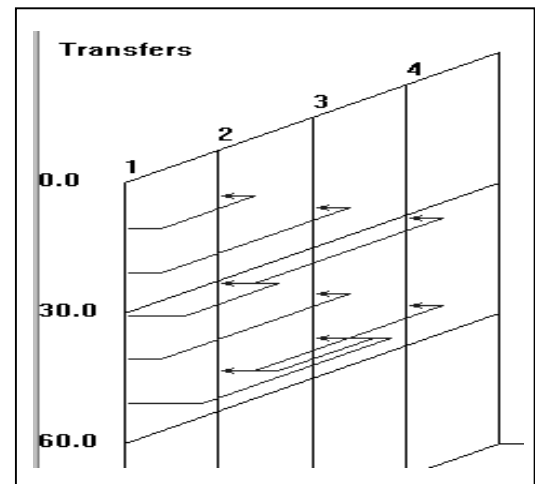


Figure 6: The Trace Viewer’s Visual Display of the First Eight Transfers in a Simulation Performed With the Model of Figure 5

The two transfers at time 30.0 in Figure 6 show an instance in which the Figure 5 model does *not* implement strict first-come, first-served service order. First the transfer from element 2 to element 4 takes place (the first user of the server finishes with the server and terminates). Then the transfer from element 1 to element 2 takes place (the *third* unit of traffic arrives and captures the server without delay). Although removed from the user chain with the intention that it should capture the server, the *second* unit

of traffic cannot make the capture, because the third unit of traffic has already done so. In effect, the third arrival “cut into line” ahead of the second arrival, so service order is not first-come, first-served in this case.

6 CONCLUSION

A class of metamodels that supports analysis of certain aspects of queuing-system simulation models has been introduced and described. Metamodels in this class are constructed by using tools presented here. These algorithmic and executable metamodels are able to reconstruct a simulation model’s behavior through analysis of trace files. Except for the need to instrument the original simulation model by equipping it with measuring points designed to produce a trace file, the tools provided are general purpose (that is, independent of the language used to build the simulation model originally). Analysis based on the methodology introduced here supports model verification on the part of interested parties (e.g., not only the builder of the model but also third parties who might be charged with the responsibility of independent model verification).

ACKNOWLEDGEMENTS

We gratefully acknowledge the assistance of Russell R. Barton, James O. Henriksen, and Robert G. Sargent, who read versions of this paper and provided comments that helped improve the paper.

REFERENCES

- Arthur, J. D., and R. E. Nance. 1996. Independent Verification and Validation: A Missing Link in Simulation Methodology? In *Proceedings of the 1996 Winter Simulation Conference*, 230-236. La Jolla, California: Society for Computer Simulation.
- Balci, O. 1995. Principles and Techniques of Simulation Validation, Verification, and Testing. In *Proceedings of the 1995 Winter Simulation Conference*, 147-154. La Jolla, California: Society for Computer Simulation.
- Banks, J. 1996. Output Analysis Capabilities of Simulation Software. *Simulation* 66 (1), (January 1996), 23-30.
- Barton, R. R. 1994. Metamodeling: a state of the art review. In *Proceedings of the 1994 Winter Simulation Conference*, 237-244. LaJolla, California: Society for Computer Simulation.
- Caughlin, D. 1997. Automating the metamodeling process. In *Proceedings of the 1997 Winter Simulation Conference*, 978-985. LaJolla, California: Society for Computer Simulation.
- Crain, R. C. 1997. Simulation using GPSS/H. In *Proceedings of the 1997 Winter Simulation Conference*, 567-573. LaJolla, California: Society for Computer Simulation.
- Frantz, F. K. 1995. A taxonomy of model abstraction techniques. In *Proceedings of the 1995 Winter Simulation Conference*, 1413-1420. LaJolla, California: Society for Computer Simulation.
- Henriksen, J. O. 1997. The power and performance of Proof Animation. In *Proceedings of the 1997 Winter Simulation Conference*, 574-580. LaJolla, California: Society for Computer Simulation.
- Huber, K. P., M. R. Berthold, and H. Szczerbicka. 1996. Fuzzy graph based metamodeling. In *Proceedings of the 1996 Winter Simulation Conference*, 418-425. La Jolla, California: Society for Computer Simulation.
- Law, A. M., and W. D. Kelton. 1991. *Simulation Modeling and Analysis*. Second Edition, New York: McGraw-Hill.
- Sargent, R. G. 1996. Verifying and validating simulation models. In *Proceedings of the 1996 Winter Simulation Conference*, 55-64. LaJolla, California: Society for Computer Simulation.
- Schriber, T. J., and D. T. Brunner. 1996. Inside simulation software: how it works and why it matters. In *Proceedings of the 1996 Winter Simulation Conference*, 23-30. LaJolla, California: Society for Computer Simulation.
- Schriber, T. J., and D. T. Brunner. 1998. How discrete-event simulation software works. Chapter 24 in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, ed. J. Banks. New York: John Wiley & Sons.
- Tolujev, J. 1997a. Werkzeuge des simulations-experten von morgen. (Tools for the simulation experts of tomorrow.) In *Simulation und Animation '97*, eds. O. Deussen and P. Lorenz 201-210. Ghent, Belgium: Society for Computer Simulation International.
- Tolujev, J. 1997b. Werkzeuge zur neutralisierung und erweiterten bearbeitung der von materialfluß-modellen erzeugten trace files. (Tools for language-independent evaluation of trace files produced by material-movement models.) In *Tagungsband 11. Symposium Simulationstechnik ASIM '97*, eds. A. Kuhn and S. Wenzel, 714-719. Braunschweig, Germany: Vieweg Press.

AUTHOR BIOGRAPHIES

JURI TOLUJEV is an Associate Professor in the Department of Modeling and Simulation at Riga Technical University in Riga, Latvia. He received his degree of Doctor of Engineering from the Riga Technical University in 1976. His research interests include simulation modeling and analysis, model diagnostics, and qualitative analysis. In 1996-97 he was a guest professor at the Institute for Simulation and Graphics at the University of Magdeburg in Magdeburg, Germany.

PETER LORENZ is a Professor at the Institute for Simulation and Graphics at the University of Magdeburg in Magdeburg, Germany, where he teaches simulation, animation, and graphics. His research interests include layout-based generation of simulation models, Web-supported delivery of simulation, and applications of simulation and animation in manufacturing, logistics and traffic.

DANIEL BEIER is a student in the Department of Computer Science, Institute for Simulation and Graphics, at the University of Magdeburg in Magdeburg, Germany. His current area of research is the interaction of simulation systems with networks, especially the World Wide Web.

THOMAS J. SCHRIBER is a Professor of Computer and Information Systems at the University of Michigan in Ann Arbor, Michigan. He is a Fellow of the Institute of Decision Sciences and is the 1996 recipient of the INFORMS College of Simulation Distinguished Service Award. He teaches and works in the area of discrete-event simulation and decision analysis.