Invited Paper – Proceedings of the 8th International Workshop on Petri Nets and Performance Models, Zaragoza, Spain, Sept. 1999.

INTEGRATED FRAMEWORKS FOR MULTI-LEVEL AND MULTI-FORMALISM MODELING[1]

William H. Sanders

Center for Reliable and High-Performance Computing
Coordinated Science Laboratory and Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA
whs@crhc.uiuc.edu
www.crhc.uiuc.edu/PERFORM

***Abstract*— There have been significant advances in methods for specifying and solving models that aim to predict the performance and dependability of computer systems and networks. At the same time, however, there have been dramatic increases in the complexity of the systems whose performance and dependability must be evaluated, and considerable increases in the expectations of analysts that use performance/dependability evaluation tools. This paper briefly reviews the progress that has been made in the development of performance/dependability evaluation tools, and argues that the next important step is the creation of modeling frameworks and software environments that support multi-level, multi-formalism modeling and multiple solution methods within a single integrated framework. In addition, this paper presents an overview of the Möbius project, which aims to provide a modeling framework and software environment that support multiple modeling formalisms, methods for model composition and connection, and a way to integrate multiple analytical/numerical- and simulation-based model solution methods. Finally, it suggests research that must take place to make this aim a reality, and thus facilitate the performance and dependability evaluation of complex computer systems and networks.**

1. Introduction

Software environments for predicting the performance, dependability, and performability of complex computer systems and networks have become widespread, and have contributed significantly to the design of such systems. The capabilities of such modeling tools have increased greatly over the last two decades, but this increase has been offset by a growth in both the complexity of systems to be analyzed and industrial users' expectations of the tools. Modern systems are complex combinations of computing hardware, networks, operating systems, and application software, and it is difficult, if not impossible, to characterize the performance and/or dependability of such systems using a single modeling formalism or single model solution technique.

These challenges call for the development of performance/dependability modeling frameworks and software environments that can predict the performance of complete distributed computing systems, accounting for all system components, including the application itself, the operating system, and the underlying computing and communication hardware. Ultimately, a framework should provide a method by which multiple, heterogeneous models can be composed together, each representing a different software or hardware module, component, or view of the system. The composition techniques developed should permit models to interact with one another by sharing state, events, or results, and should be scalable, in the sense that the solution of an entire model should be possible at a cost lower than for an equivalent unstructured model. A framework should also support multiple modeling languages, as well as methods to combine models at different levels of resolution. Furthermore, a framework should support multiple model solution methods, including both simulation and analysis, that are efficient, and permit the solution of complete models of complex computing and communication systems, and the applications executing on such systems. Finally, a framework should be extensible, in the sense that it should be possible to add new modeling formalisms, composition and connection methods, and model solution techniques to a software environment that implements the framework without changing existing tool components.

2. The State of the Art

While the requirements listed in the preceding section may seem well beyond the capability of current tools, dramatic progress has been made toward frameworks that have these capabilities, suggesting that it may now be possible to develop a framework and software environment that meet many of these goals. One major advance was the development of tools that provided a single high-level formalism for specifying models, and provided multiple ways in which a model expressed in that formalism could be solved, depending on the characteristics of the particular model. While these tools do not support the multiple model specification and composition methods needed in a complete modeling framework, they often implement multiple solution methods, recognizing the fact that no single solution method is sufficient for all models.

One set of tools in this category is those that use some form of queuing networks as their specification method. These tools include DyQN-Tool+ [Haverkort 95], which uses dynamic queueing networks as its model specification method; HIT [Beilner 88], which uses a homogeneous, structured paradigm for specifying systems; LQNS [Franks

[1] This research has been supported, in part, by DARPA Contract DABT63-96-C-0069 and Motorola Satellite Communications.

95], which uses layered queuing networks as its specification language; QNAP2 [Veran 85], which allows users to specify a model as a network of service stations; and RESQ and RESQME [Chang 93], which use extended queueing networks as their specification method. In most cases, these tools support both simulation- and product-form-based solutions.

Another set of tools in this category is those based on stochastic Petri nets and extensions. There are many tools in this category; for a comprehensive list, see [Aarhus, Trompedeller]. These tools include DSPNexpress [Lindemann 99], GreatSPN [Chiola 95], QPN-Tool and HiQPN-Tool [Bause 95a], SPNP [Ciardo 93], SPNL [German 97], SURF-2 [Béounes 93], TimeNET [German 95], and *Ultra-SAN* [Sanders 95], among others. In each case, the tool supports model specification using some, possibly hierarchical, variant of stochastic Petri nets, and provides analytic/numerical solution of a generated state-level representation. In some cases, the tools support simulation-based solution as well.

Finally, there are a number of tools in this category that use other model specification approaches, sometimes tailored to a particular application domain. These tools include DEPEND [Goswami 97], Figaro [Bouissou 93], HARP [Bavuso 87], HIMAP [Sridharan 98], and SAVE [Goyal 86], which all focus on evaluating the dependability of fault-tolerant computing systems. They also include SPE•ED^{TM} [Smith 97], which aims to aid in software performance engineering, and TANGRAM-II [Carmo 97], which evaluates computer and communication systems using analytical/numerical methods.

While each of these tools is useful for its intended application and within the range of solutions supported by the particular solution method(s) implemented, none of them, individually, meet the goals for an integrated performance/dependability modeling framework that was outlined earlier. Furthermore, extending any of these tools to meet those goals would be difficult, since they were all built with a particular modeling formalism and solution technique or set of solution techniques in mind, rather than with the aim of extensibility. New performance/dependability modeling frameworks and software environments are thus needed, if we are to succeed in evaluating modern-day computer systems and networks. Two approaches have been taken in this regard.

In the first, a software environment is created that facilitates the combination of multiple tools of the type just described into a single environment. A perspective on this idea, in the context of software performance engineering, can be found in [Smith 91]. We are aware of three tools that take this approach. The first, called IMSE (Integrated Modeling Support Environment) [Pooley 91], is a support environment for performance modelers that contains tools for modeling, workload analysis, and system specification. The second, called IDEAS (Integrated Design Environment for Assessment of Computer Systems and Communication Networks) [Fricks 96, Fricks 97], aims to give an analyst a broad range of modeling capabilities without the need to learn multiple interface languages and output formats. The third, called Freud [van Moorsel 98], has aims similar to those of IMSE and IDEAS, but focuses on providing a uniform interface to a variety of web-enabled tools.

In certain of these tools, the primary focus is on providing both a common graphical interface by which a user accesses each constituent tool, and a common method for reporting results. In others, the most important feature is a method by which results that are obtained using one tool can be used as input values in another tool. While these approaches are important steps toward building an integrated software environment of the type we described earlier, they are inherently limited in the way models expressed in one tool can interact with those expressed in another tool, since they can only exchange information via output from the individual tools. Furthermore, the degree to which the user interfaces of the constituent tools can be integrated depends on their similarity. In short, building an integrated performance/dependability modeling environment by combining existing modeling tools provides some of the features that we believe are needed but, because the tools that are combined were not designed to be integrated, greatly limits the degree to which models of different parts of a system can interact.

The second approach toward building an integrated performance/dependability modeling environment is to start from scratch, and define a modeling framework that can accommodate multiple modeling formalisms, multiple ways to combine models expressed in different formalisms, and multiple model solution methods. Though more difficult than building a software environment out of existing tools, this approach has the potential to much more closely integrate models expressed in different modeling formalisms, while preserving and exploiting the structure of particular modeling formalisms within the framework.

The earliest attempt to do this, to the best of our knowledge, is the combination of multiple modeling formalisms in the SHARPE modeling tool [Sahner 86, Sahner 96]. In the SHARPE modeling framework, models can be expressed as combinatorial reliability models, directed acyclic task precedence graphs, Markov and semi-Markov models, product-form queueing networks, and generalized stochastic Petri nets. Models expressed in the framework can be solved by exchanging results expressed as exponential-polynomial distribution functions between submodels. SHARPE is thus an important step forward in the development of an integrated performance/dependability modeling environment, in that it obtains solutions to models expressed in multiple formalisms by exchanging results.

Another software environment that integrates multiple modeling formalisms in a single software environment is SMART [Ciardo 96, Ciardo 97]. SMART supports the analysis of models expressed as stochastic Petri nets and queueing networks, and is implemented in a way that permits the easy integration of new solution algorithms in the tool. The interface to the tool is textual, and models expressed in possibly different formalisms can be combined by exchanging results, possibly repeatedly, using fixed-point iteration. Like SHARPE, SMART is an important

step forward in that it permits the solution of models made up of multiple submodels that exchange results.

The DEDS (Discrete Event Dynamic System) toolbox [Bause 95, Bause 98] also integrates multiple modeling formalisms into a single software environment, but does so in a manner very different from the previous two tools. In particular, the DEDS toolbox converts models expressed in different modeling formalisms (including queueing networks, generalized stochastic Petri nets, and colored Petri nets) into a common "abstract Petri net notation" [Bause 95, Bause 95b]. Once expressed in this formalism, models can be solved using a variety of functional and quantitative analysis approaches for Markovian models.

Each of these projects showed that it is possible to build a modeling framework and software environment that integrate models expressed in multiple formalisms more closely than would be possible if one integrated existing tools. In the case of SHARPE and SMART, composite models are built by exchanging results obtained by solving possibly heterogeneous submodels; in the case of the DEDS toolbox, the integration is obtained by converting models expressed in different formalisms into a common abstract notation. The next step in generality is to build a modeling framework without presupposing what types of modeling formalisms would be supported or what methods would be used to combine submodels. If this could be done, we would succeed in building a tool that supports the evaluation of complex computer systems and networks and is also extensible. While this goal is probably impossible to achieve completely in practice (since certain assumptions must be made in implementing a conceptual framework), it guided us in the development of the Möbius modeling framework, which we describe in the next section.

3. The Möbius Modeling Framework

The Möbius modeling framework, as illustrated in Figure 1, facilitates the development of multiple modeling languages (at different levels of detail and abstraction), multiple model composition methods, multiple performance measure specification methods, multiple model connection methods (here, "connection" denotes the exchange of performance measures between models), and multiple model solution methods. A key part of the design of Möbius is an object-oriented approach that permits multiple models expressed in heterogeneous modeling languages to interact with multiple model solution methods without the solution methods or models themselves knowing the semantics of each modeling language.

To permit this, an "abstract functional interface" was developed [Doyle 99]. The *abstract functional interface* specifies a set of methods that each modeling formalism defined in the framework must implement, and that can be used by models expressed in multiple modeling formalisms to interact with each other and with multiple model solution methods. For example, one of these method interfaces specifies a way to share the "state" of a model expressed in each language, so events in models expressed in different languages can know and affect each other's states. Furthermore, method interfaces specify how model solution methods interact with formalisms to execute them, in the case of simulation [Williamson 98], or to generate the possible states they can reach, in the case of state-based solution methods [Sowder 98]. By carefully defining a set of "base methods" by which multiple models and model solution engines interact with one another, the Möbius modeling framework is extensible, in the sense that new modeling languages at multiple levels of abstraction and new analytical/numerical- and simulation-based solution methods can be added to a tool that is implemented using the framework without changing existing tool components.

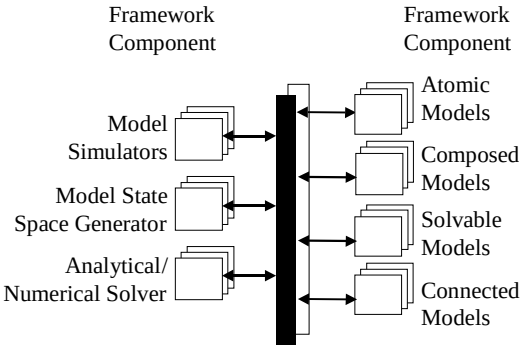

Figure 1: Möbius Modeling Framework

More specifically, as seen in Figure 1, the Möbius framework supports four different model language types, and multiple (heterogeneous) languages can be defined for each type. "Atomic models" are the first model type. *Atomic models* are defined by a set of *actions*, *state variables*, and the rules describing how the completion of actions changes the state of the model. The state variables themselves are typed (or, in other words, have a precisely defined structure), and knowledge of their type permits the framework to specify which parts of the state in one model can be associated with certain state variable(s) in another model [Doyle 99].

A composed model [Stillman 99] specifies a function that takes one or more models (atomic or composed) as input and produces a model as its output. Informally, a composed model specification language allows one to build heterogeneous, hierarchical models out of multiple submodels. It is important to note that a composed model specification language does not perform its function by reducing all the input models to a common, "universal" language (such as is done in the DEDS toolkit, for example). Instead, by specifying how models interact with one another using the set of base methods described earlier, composed model specification languages both preserve the structure of a model's constituent submodels and expose new structure (by specifying precisely how the submodels interact with one another).

This preservation of structure is extremely important in building hierarchical models that can be solved efficiently, since it allows special properties of their structure to aid in

the solution process. For example, the "Replicate/Join" model composition language [Sanders 91] uses the structure of the composed model to identify and exploit symmetries that are present in a model, dramatically reducing the size state space that must be generated [Sanders 91], or speeding future event list management [Sanders 93]. More generally, "composed model graphs" [Obal 98] permit the identification of all types of symmetries that are present in a composed model structure.

Reward variables [Kavanaugh 98] specify a set of performance variables to be calculated from a model. Generally speaking, they permit one to specify "rates" of reward (either positive or negative) that accumulate while a model is in a state, and "impulses" of reward that are obtained upon completion of an action. In addition, reward variables can have state, whose value can be changed by actions that complete in the model on which a variable is defined. As with the previous model types, multiple reward model specification languages can be defined within the Möbius framework.

A *solvable model* is built from one or more atomic or composed models and a set of reward variables. A solvable model is "solvable" in the sense that it prescribes both a model that can be executed, and a set of measures to collect from the model.

Finally, one or more solvable models can be combined to form a *connected model*. Connected models interact with one another by sharing results, as defined by the reward structure of the constituent reward models. Thus, like composed models, they support the construction of hierarchical models, this time in a more loosely coupled way, in which the component models interact by exchanging results. The results can be means, variances, or probability distribution functions of either particular reward variables or, more generally, more abstract models constructed by solving a particular component model. The exchange of exponential-polynomial distributions between submodels in SHARPE [Sahner 96] is one example of a connected model class; another example would be a modeling language that supports fixed-point iteration.

Figure 2 shows the interaction between these model types, and how one model type can be constructed from another. Note that composed models can be constructed from one or more atomic or composed models; solvable models can be constructed from a set of reward variables and an atomic, composed, or solvable model; and connected models can be constructed from a set of reward variables and one or more solvable models, or from a set of reward variables and a connected model. As described earlier, the use of a well-defined set of base classes permits models to interact with one another without knowing the details of the modeling language in which they are implemented. This makes it easy to add new modeling formalisms, composition and connection methods, and solution methods to a tool implemented in the Möbius framework.

4. Prototype Software Environment

The utility of the Möbius framework has been demonstrated by implementation of an example atomic model language, a composed model language, and a performance variable specification method [Deavours 99, Deavours 99a]. The software environment that we have developed permits the inclusion of new model formalism editors and model solution engines as they are developed. Within this environment, we have implemented an atomic model editor [Doyle 99] based on stochastic activity networks [Meyer 85, Sanders 88], a composed model editor [Stillman 99] that implements the replicate/join model composition language [Sanders 91], a composed model editor [Stillman 99] that implements the graph composition formalism in [Obal 98], and a SAN-based reward variable specification language [Sanders 91a]. Figure 3 shows the Möbius project manager interface. Note that the tabs on the panel correspond to different model types within the framework. In each case, when a user creates a new model, he/she is asked to select from among the allowable modeling formalisms within that model's type.

While most of these modeling languages and techniques have been previously implemented in the modeling tool known as *UltraSAN* [Sanders 95], their implementation in the Möbius framework showed the ability of the use of base classes to hide modeling-language-specific details from models that interact with one another.

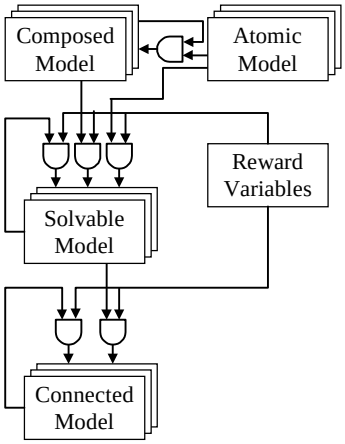

Figure 2: Model Type Interactions in Möbius

Moreover, the act of implementing multiple non-trivial model specification languages using the Möbius abstract functional interface aided us in precisely defining the base classes themselves, by helping us determine how multiple modeling languages should best interact with one another with minimal knowledge of each other's internals. Finally, the implementation allowed us to find, through experimentation, an implementation of the base classes that hides as many details as possible of individual modeling formalisms while still providing good performance in the result-

ing tool. In particular, we were able to obtain simulation model solutions approximately 50% faster [Williamson 98], and generate state spaces slightly faster and with approximately 4 times less memory [Sowder 98], than our previous *UltraSAN* implementation for our initial benchmark applications. These results bode well for using the Möbius framework to build software environments with multiple model specification methods at differing levels of detail and abstraction, multiple model composition and connection methods, multiple performance variable specification methods, and multiple model solution methods.

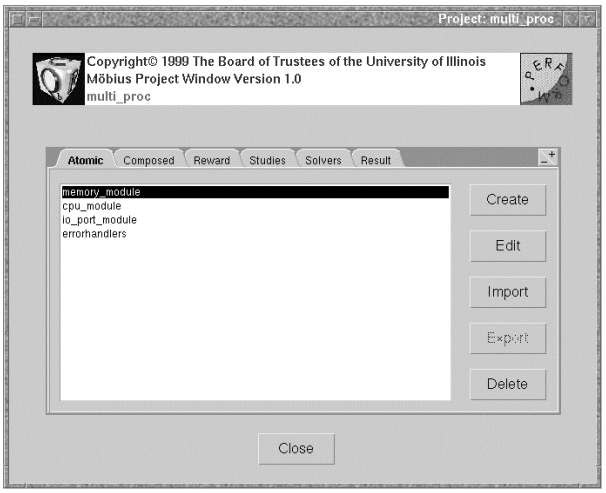

Figure 3: Möbius Project Manager Interface

5. Future Research Directions

While Möbius provides a basis for developing multiple model formalisms, multiple composition and connection methods, and multiple model solution methods in a single, integrated framework, it does not prescribe the particular formalisms, composition, connection methods, and solution methods provided in a particular software environment that uses the framework. Indeed, one goal of the framework was to facilitate modeling research by making it possible to test new techniques by implementing them in the framework, rather than in a stand-alone environment. With this in mind, we suggest several avenues of research that are needed to make multi-formalism, multi-level modeling tools a reality:

Domain-specific and domain-independent modeling languages for computing hardware, networks, operating systems, and application software

In order to represent parts of a computer system/network in a model accurately, both domain-specific and domain-independent modeling languages must be developed. While academics and researchers may be satisfied with abstract modeling languages such as stochastic Petri nets, we believe that many models developed in industry would be better expressed in domain-specific modeling languages that are tailored to particular application domains. Models expressed in these languages could be generated using graphical user interfaces in which a system designer can directly express possible system designs, or could be generated automatically by a software/hardware design tool. Since no single level of abstraction will be appropriate for all systems at all stages of their development, multiple model description languages should be developed, and an investigation should be made of how parts of a system, described at different levels of abstraction and detail, can interact with one another. In addition, domain-specific languages should be developed for specifying the wide variety of performance/dependability measures of interest to system developers. These should include standard measures, such as response time, throughput, reliability, availability, and probability of meeting prescribed deadlines, as well as application-specific measures specified by an application designer. The Möbius reward variable framework makes it easy to do this in a model-language-independent way.

Composition methods that use the structure of models in their solution

New composition methods that use the structure of models in their solution should be investigated. More specifically, finding ways to use the structure of a model expressed in a particular composition formalism is necessary if we are to avoid the explosive growth in solution time and complexity that would come if component models were naively composed to form a complete system model. One promising direction here is that of exploiting the symmetries inherently present in parallel code and system architectures to aid in solution of complete system models.

Connection methods that are exact, or that give an estimate of the error they induce through their use

Complementary to the composition methods just discussed, new methods should be developed that can be used to connect together models of different parts of a complete system by exchanging "results." In this context, results may be the mean, moments, or distribution of a performance/dependability variable specified in a solvable model, or a "higher-level" model derived from a solvable model. Examples of higher-level models that could be derived from a solvable model include hidden Markov models, order statistics, and Markov-modulated Poisson processes.

In special cases, connection models can produce exact results, but most often they will produce results that are approximate, since the results exchanged by models will be a "condensed" characterization of their behavior. For example, one could develop a connection method for processors that replaces the execution of a basic block of code with a single mean execution time – and in doing so, lose some of the precision that would be obtained by a more detailed model. To quantify this loss in precision, methods should be developed that can estimate (or bound) the error that occurs when such connections are made. Such connection methods, together with the composition methods described in the previous subsection, would aid in the construction of system models that are scalable, in the sense that their so-

lution complexity would grow with system size growth at a manageable rate.

Automated methods for incorporating measurement results in developed models

In addition to developing modeling methods per se, research is needed to integrate modeling frameworks with results from measurements, which can be used to better determine model parameter values. This could be done by developing a common data exchange format for measurement, in which the particular measurements collected would be dictated by the input parameters needed by the particular system under study.

6. Conclusions

In summary, integrated modeling frameworks are needed that can be used to predict the performance of applications written for a wide variety of computer systems, including networks of workstations, clusters, or larger-scale networked systems. They should be usable in a stand-alone fashion, or in conjunction with measurement results provided by a prototype or deployed implementation. Furthermore, software environments implemented using a framework should be extensible, in the sense that new modeling languages, composition and connection methods and languages, and model solution methods can be added without changing modules already implemented. In this manner, tools implemented using the framework would support hierarchical, multi-level modeling of complete systems, including applications, and would be scalable through the innovative use of model composition and connection methods.

The Möbius framework is one step in this direction, and attempts to facilitate progress toward these goals by taking an object-oriented approach that permits multiple models expressed in heterogeneous modeling languages to interact with multiple solution methods without the solution methods or models themselves knowing the semantics of each modeling language.

Work on the Möbius framework and its associated software environment is ongoing. While the work to date has been done at the University of Illinois, we intend to publish specifications of the method interfaces Möbius uses so that others may develop new atomic model formalisms, model composition methods, model connection methods, and model solution methods. Since the Möbius tool is extensible, others should be able to contribute new modeling formalisms, composition and connection methods, and model solution methods that will work with current tool components without requiring them to change. It is our hope that this work will result in performance/dependability modeling tools that are useful to industry and provide a research infrastructure that facilitates development of new modeling techniques and solution algorithms.

Acknowledgements

The author would like to thank the entire Möbius team for their hard work and dedication, which made these ideas a reality: David Daly, Dan Deavours, Jay Doyle, G. P. Kavanaugh, John Sowder, Aaron Stillman, Patrick Webster, and Alex Williamson. He would also like to thank the Program Chair, Peter Buchholz, and the Conference Committee for inviting him to present this paper at the PNPM'99 Workshop.

References

[Aarhus] S. Christensen and K. H. Mortensen, maintainers, "Tools on the Web" web site at the Computer Science Department, University of Aarhus (http://www.daimi.au.dk/~petrinet/tools/).

[Bause 95] F. Bause and H. Beilner (eds.), *Performance Tools: Model Interchange Formats, Forschungsbericht Nr. 581 des Fachbereichs Informatik der Universität Dortmund*, Germany, 1995.

[Bause 95a] F. Bause, P. Buchholz, and P. Kemper, "QPN-tool for the Specification and Analysis of Hierarchically Combined Queueing Petri Nets," in *Quantitative Evaluation of Computing and Communication Systems: Proc. of the 8th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation (Performance Tools '95) and the 8th GI/ITG Conf. on Measurement, Modelling and Evaluating Computing and Communication Systems (MMB '95)*, Heidelberg, Germany, September 1995. *Lecture Notes in Computer Science*, no. 977 (ed. by H. Beilner and F. Bause), Berlin: Springer, 1995, pp. 224-238

[Bause 95b] F. Bause, P. Kemper, and P. Kritzinger, "Abstract Petri Net Notation," *Petri Net Newsletter*, no. 49, pp. 9-27, 1995.

[Bause 98] F. Bause, P. Buchholz, and P. Kemper, "A Toolbox for Functional and Quantitative Analysis of DEDS," in *Computer Performance Evaluation: Modelling Techniques and Tools: Proc. of the 10th Int. Conf., Tools '98*, Palma de Mallorca, Spain, Sept. 14-18, 1998, *Lecture Notes in Computer Science* No. 1469 (ed. by R. Puigjaner, N. N. Savino, and B. Serra), Berlin: Springer, 1998, pp. 356-359.

[Bavuso 87] S. J. Bavuso, J. Bechta Dugan, K. S. Trivedi, E. M. Rothmann, and W. E. Smith, "Analysis of Typical Fault-Tolerant Architectures Using HARP," *IEEE Trans. on Reliability*, vol. 36, no. 2, pp. 176-185, 1987.

[Beilner 88] H. Beilner, J. Mäter, and N. Weißenberg, "Towards a Performance Modelling Environment: News on HIT," in *Proc. of the 4th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Plenum Publishing, 1988.

[Béounes 93] C. Béounes et al., "SURF-2: A Program for Dependability Evaluation of Complex Hardware and Software Systems," in *23rd Int. Symp. on Fault-Tolerant Computing*, Toulouse, France, 1993, pp. 668-673.

[Bouissou 93] M. Bouissou, "The FIGARO Dependability Evaluation Workbench in Use: Case Studies for Fault-Tolerant Computer Systems," in *Proc. of the 23rd Annual Int. Symp. on Fault-Tolerant Computing*, Toulouse, France, June 1993, pp. 680-685.

[Carmo 97] R. M. L. R. Carmo, L. R. de Carvalho, E. de Souza e Silva, M. C. Diniz, and R. R. R. Muntz, "TANGRAM-II: A Performability Modeling Environment Tool," in *Computer Performance Evaluation: Modelling Techniques and Tools: Proc. of the 9th Int. Conf.*, St. Malo, France, June 3-6, 1997, *Lecture Notes in Computer Science*, No. 1245 (ed. by R. Marie, B. Plateau, M. Calzarossa, and G. Rubino). Berlin: Springer, 1997, pp. 6-18.

[Chang 93] K. C. Chang, R. F. Gordon, P. G. Loewner, and E. A. MacNair, "The RESearch Queueing Package Modeling Environment (RESQME)," IBM Research Report RC-18687, Yorktown Heights, New York, Feb. 1993.

[Chiola 95] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo, "GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets," in *Performance Evaluation*, vol. 24, no. 1-2, pp. 47-68, Nov. 1995.

[Ciardo 93] G. Ciardo and K. S. Trivedi, "SPNP: The Stochastic Petri Net Package (Version 3.1)," in *Proc. 1st Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'93)*, San Diego, CA, USA, Jan. 1993, pp. 390-391.

[Ciardo 96] G. Ciardo and A. S. Miner, "SMART: Simulation and Markovian Analyzer for Reliability and Timing," in *Proc. IEEE Int. Computer Performance and Dependability Symp. (IPDS'96)*, Urbana-Champaign, IL, USA, Sept. 1996, pp. 60.

[Ciardo 97] G. Ciardo and A. S. Miner, "SMART: Simulation and Markovian Analyzer for Reliability and Timing," in *Tools Descriptions from the 9th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation and the 7th Int. Workshop on Petri Nets and Performance Models*, Saint Malo, France, June 1997, pp. 41-43.

[Deavours 99] D. Daly, D. D. Deavours, J. M. Doyle, A. J. Stillman, and P. G. Webster, "Möbius: An Extensible Tool for Performance and Dependability Modeling," in *Digest of FastAbstracts presented at the 29th Annual Int. Symp. on Fault-Tolerant Computing (FTCS-29)*, Madison, Wisconsin, USA, June 15-18, 1999, pp. 15-16.

[Deavours 99a] D. Daly, D. D. Deavours, J. M. Doyle, A. J. Stillman, P. G. Webster, and W. H. Sanders, "Möbius: An Extensible Framework for Performance and Dependability Modeling," in *Tool Descriptions of the 8th Int. Workshop on Petri Nets and Performance Models*, Zaragoza, Spain, Sept. 1999.

[Doyle 99] J. M. Doyle, "Abstract Model Specification Using The Möbius Modeling Tool," Master's Thesis, University of Illinois, 1999.

[Franks 95] R. G. Franks, A. Hubbard, S. Majumdar, J. E. Neilson, D. C. Petriu, J. Rolia, and C. M. Woodside, "A Toolset for Performance Engineering and Software Design of Client-Server Systems," *Performance Evaluation*, vol. 24, no. 1-2, pp. 117-135, Nov. 1995.

[Fricks 96] R. Fricks, S. Hunter, S. Garg, and K. Trivedi, "IDEA: Integrated Design Environment for Assessment of ATM Networks," in *Proc. Second IEEE Int. Conf. on Engineering of Complex Computer Systems*, Montreal, Canada, Oct. 21-25, 1996, pages 27-34.

[Fricks 97] R. Fricks, C. Hirel, S. Wells, and K. Trivedi, "The Development of an Integrated Modeling Environment," in *Proc. World Congress on Systems Simulation (WCSS '97), (2nd Joint Conf. of Int. Simulation Societies)*, Singapore, Sept. 1-3, 1997, pp. 471-476.

[German 95] R. German, C. Kelling, A. Zimmermann, and G. Hommel, "TimeNET: A Toolkit for Evaluating Non-Markovian Stochastic Petri-Nets," *Performance Evaluation*, vol. 24, pp. 69-87, 1995.

[German 97] R. German, "SPNL: Processes as Language-oriented Building Blocks of Stochastic Petri Nets," in *Computer Performance Evaluation, Lecture Notes in Computer Science*, No. 1245, Berlin: Springer, 1997, pp. 123-134.

[Goswami 97] K. K. Goswami and R. K. Iyer, "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis," *IEEE Trans. on Computers*, vol. 46, no. 1, pp. 60-74, Jan. 1997.

[Goyal 86] A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg, and K. S. Trivedi, "The System Availability Estimator," in *Proc. of the 16th Int. Symp. on Fault-Tolerant Computing (FTCS-16)*, 1986, pp. 84-89.

[Haverkort 95] B. R. Haverkort, "Performability Evaluation of Fault-Tolerant Computer Systems Using DyQNtool$^+$," *Int. Journal of Reliability, Quality, and Safety Engineering*, vol. 2, no. 4, pp. 383-404, 1995.

[Kavanaugh 98] G. P. Kavanaugh III, "Design and Implementation of an Extensible Tool for Performance and Dependability Model Evaluation," Master's Thesis, University of Illinois, 1998.

[Lindemann 99] C. Lindemann, A. Reuys, and A. Thümmler, "The DSPNexpress 2.000 Performance and Dependability Modeling Environment," in *Proc. of the 29th Annual Int. Symp. on Fault-Tolerant Computing*, Madison, Wisconsin, USA, June 15-18, 1999, pp. 228-231.

[Meyer 85] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic Activity Networks: Structure, Behavior, and Application," in *Proc. of the Int. Conf. On Timed Petri Nets*, Torino, Italy, July 1995, pp. 106-115.

[Obal 98] D. Obal, "Measure-Adaptive State-Space Construction Methods," Doctoral Dissertation, University of Arizona, 1998.

[Pooley 91] R. J. Pooley, "The Integrated Modelling Support Environment: A New Generation of Performance Modelling Tools," in *Computer Performance Evaluation: Modelling Techniques and Tools: Proc. of the Fifth Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Torino, Italy, February 13-15,

1991 (ed. by G. Balbo and G. Serazzi), Amsterdam: Elsevier, 1992, pp. 1-15.

[Sahner 86] R. A. Sahner, "Combinatorial-Markov Method of Solving Performance and Reliability Models," Ph.D. Dissertation, Duke University, 1986.

[Sahner 96] R. A. Sahner, K. S. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems, An Example-Based Approach Using the SHARPE Software Package*, Boston, MA: Kluwer, 1996.

[Sanders 88] W. H. Sanders, "Construction and Solution of Performability Models Based on Stochastic Activity Networks," Doctoral Dissertation, University of Michigan, 1988.

[Sanders 91] W. H. Sanders and J. F. Meyer, "Reduced Base Model Construction Methods for Stochastic Activity Networks," *IEEE Journal on Selected Areas in Communications, special issue on Computer-Aided Modeling, Analysis, and Design of Communication Networks*, vol. 9, no. 1, Jan. 1991, pp. 25-36.

[Sanders 91a] W. H. Sanders and J. F. Meyer, "A Unified Approach for Specifying Measures of Performance, Dependability, and Performability," in *Dependable Computing for Critical Applications, Vol. 4 of Dependable Computing and Fault-Tolerant Systems* (ed. A. Avizienis, H. Kopetz, and J. Laprie), Springer-Verlag, 1991, pp. 215-237.

[Sanders 93] W. H. Sanders and R. S. Freire, "Efficient Simulation of Hierarchical Stochastic Activity Network Models," in *Discrete Event Dynamic Systems: Theory and Applications*, vol. 3, no. 2/3, July 1993, pp. 271-300.

[Sanders 95] W. H. Sanders, W. D. Obal II, M. A. Qureshi, and F. K. Widjanarko, "The *UltraSAN* Modeling Environment," in *Performance Evaluation*, vol. 24, no. 1, Oct.-Nov. 1995, pp. 89-115.

[Smith 91] C. U. Smith, "Integrating New and 'Used' Modeling Tools for Performance Engineering," in *Computer Performance Evaluation: Modelling Techniques and Tools: Proc. of the 5th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Torino, Italy, 13-15 February 1991 (ed. by G. Balbo and G. Serazzi), Amsterdam: Elsevier, 1992, pp. 153-163.

[Smith 97] C. U. Smith and L. G. Williams, "Performance Engineering Evaluation of Object-Oriented Systems with SPE•ED™," *Computer Performance Evaluation: Modelling Techniques and Tools: Proc. of the 9th Int. Conf.*, St. Malo, France, June 3-6, 1997, *Lecture Notes in Computer Science* No. 1245 (ed. by R. Marie, B. Plateau, M. Calzarossa, and G. Rubino), Berlin: Springer, 1997, pp. 135-154.

[Sowder 98] J. M. Sowder, "State-Space Generation Techniques in the Möbius Modeling Framework," Master's Thesis, University of Illinois, 1998.

[Sridharan 98] M. Sridharan, S. Ramasubramanian, and A. K. Somani, "HIMAP: Architecture, Features, and Hierarchical Model Specification Techniques," in *Computer Performance Evaluation: Modelling Techniques and Tools: Proc. of the 10th Int. Conf., Tools '98*, Palma de Mallorca, Spain, September 14-18, 1998, *Lecture Notes in Computer Science* No. 1469 (ed. by R. Puigjaner, N. N. Savino, and B. Serra), Berlin: Springer, 1998, pp. 348-351.

[Stillman 99] A. Stillman, "Model Composition in the Möbius Modeling Framework," Master's Thesis, University of Illinois, 1999.

[Trompedeller] M. Trompedeller, maintainer, "A Petri Net Classification and Related Tools" web site (http://www.dsi.unimi.it/Users/Tesi/trompede/petri/home.html).

[van Moorsel 98] A. P. A. van Moorsel and Y. Huang, "Reusable Software Components for Performability Tools and Their Utilization for Web-based Configurable Tools," in *Computer Performance Evaluation: Lecture Notes in Computer Science* No. 1469, Berlin: Springer, 1998, pp. 37-50.

[Veran 85] M. Veran, D. Potier, "QNAP2: A Portable Environment for Queuing System Modeling," in *Modeling Techniques and Tools for Computer Performance Evaluation* (ed. by D. Potier), North-Holland, 1985, pp. 25-63.

[Williamson 98] A. L. Williamson, "Discrete Event Simulation in the Möbius Modeling Framework," Master's Thesis, University of Illinois, 1998.