Möbius 2.3: An Extensible Tool for Dependability, Security, and Performance Evaluation of Large and Complex System Models

Tod Courtney, Shravan Gaonkar, Ken Keefe, Eric W. D. Rozier, and William H. Sanders
Coordinated Science Laboratory, Information Trust Institute,
Department of Electrical and Computer Engineering and Department of Computer Science
University of Illinois at Urbana-Champaign
{tod, gaonkar, kjkeefe, ewdr, whs}@crhc.illinois.edu

Abstract

Möbius 2.3 is an extensible dependability, security, and performance modeling environment for large-scale discrete-event systems. It provides multiple model formalisms and solution techniques, facilitating the representation of each part of a system in the formalism that is most appropriate for it, and the application of the solution method or methods best-suited to estimating the system's behavior. Since its initial release in 2001, many advances have been made in Möbius' design and implementation that have strengthened its place in the modeling and analysis community. With almost a decade of widespread academic and industrial use, Möbius has proven itself to be useful in a wide variety of modeling situations. This paper documents the current feature set of Möbius 2.3, emphasizing recent significant enhancements.

1. Introduction

The most significant challenge in building practical system-level dependability modeling tools has been dealing with largeness and complexity in the systems that are modeled. Many modeling tools have been developed that can easily represent, solve, and analyze toy models or highly abstracted system representations. However, many of these modeling tools have designs that inhibit their effectiveness as the size or complexity of the models increases. The Möbius modeling tool seeks to address these issues. Möbius was first introduced in [23] with the goal of providing a flexible, extensible, and efficient framework for implementing algorithms to model and solve discrete-event systems. These goals have driven all design decisions in Möbius, and motivated us, over the last decade, to create a toolkit that supports a diverse set of modeling formalisms and solution methods.

To support these multiple modeling formalisms and solution methods, Möbius implements an *abstract functional interface* (AFI), allowing models to interact with one another without knowing the details of their implementation. Two distinct AFIs exist in Möbius: a "model-level" AFI that specifies how models change state when events occur, independent of the type of formalism in which the model is expressed, and a "state-level" AFI that allows access to model states and state transition information without knowledge of how a stochastic process (typically, a Markov process) is implemented. Using the multiple modeling formalisms implemented using the model-level AFI, a Möbius user can choose the best formalism for each component in his or her system [9]. The model-level AFI allows the component models to synchronize events or share state with other component models, even when represented using different formalisms. Similarly, the state-level AFI allows numerical solution methods to communicate with state spaces stored in a variety of representations, including sparse-matrix, Kronecker, and matrix-diagram representation [12].

This paper documents the enhancements that have been made to Möbius since the original conference [7] and subsequent journal [9] papers describing it were published. These enhancements are significant, and include new ways to build models, solve models, and analyze results that are generated. We show how the added features contribute to Möbius' ability to efficiently solve large and complex models. New symbolic state-space generators, which generate lumped Markov process representations from composed model descriptions, have been added. Another addition is a new simulation solver called the *simultaneous simulator*, which can solve families of similar models together more efficiently than if they were solved individually. Finally, several new tools (a database interface, simulation data manager, trace analyzer, and design of experiments tool) were created to make it easier to analyze results from model runs, and intelligently explore the design space represented by a parameterized set of models.

Together, these enhancements have resulted in a mature

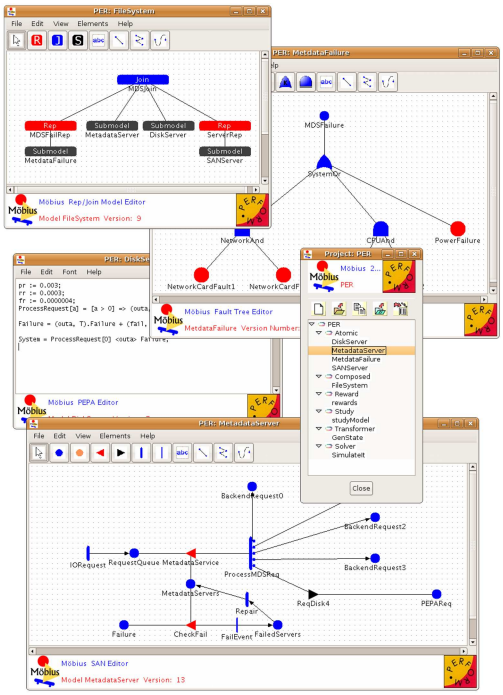

Figure 1. Example Möbius Model Editors

dependability, security, and performance modeling tool that is now widely used in academia and industry. In the remainder of this tool paper, we describe the architecture, implementation, and usage of version 2.3 of Möbius.

2. The Möbius Framework

The Möbius project was inspired by its predecessor, *UltraSAN* [25], a tool developed for modeling and solving *stochastic activity networks* (SANs [24]). Möbius was designed to exceed the capabilities of *UltraSAN* by providing an extensible, multi-formalism, multi-solution framework. The architecture of the Möbius tool was designed to allow new formalisms and solution techniques to be integrated into an underlying framework [23, 9, 12] that specifies modeling components and their interactions in a well-defined way. This is done through Möbius' model-level AFI [8].

A user begins by creating a representation of the system through a combination of atomic and composed models (see Figure 1). The model-level AFI consists of a number of abstract classes from which new modeling formalisms can be defined. The AFI treats all models as sets of two components: state variables which hold the state of the model, and actions which change the state of the model. The AFI also preserves an additional set of characteristics which may be utilized in composition or solution, such as the presence of only exponentially distributed activities. Using this set of information about a model, the Möbius architecture constructs C++ source code, and compiles it into libraries that are later linked with other generated libraries as the user proceeds down the workflow. Because the model-level AFI requires atomic and composed formalisms to adhere to its underlying structure, it is very easy for a heterogenous mix of atomic and composed models to interface. This allows the Möbius user to represent the system using any combination of atomic or composed model formalisms.

After the modeler has represented the system to be studied in Möbius he or she defines a set of reward variables that take readings from the system at instants of time, over intervals of time, over a time-averaged interval, or once the system reaches a steady state. Next, the modeler defines a range, set, or design of experiments study to vary any desired parameter values in the system during later evaluation. At this point, Möbius has generated several custom C++ libraries based on the specifications the modeler has provided. The next steps will bring those libraries together, along with many built-in Möbius libraries, to solve for the reward variables in the model.

Standard discrete-event simulation links the atomic, composed, reward, and study libraries together with the Möbius simulation libraries to create a binary that executes the simulation. Because the model-level AFI standardizes the library code expected from earlier steps, the same libraries are used in analytical solutions. First, the state space must be generated, and Möbius provides two such generators. The flat state space generator is the standard option, but a recent addition to Möbius is the symbolic state space generator, which uses state lumping techniques to drastically reduce the state space size in some models. The state-level AFI [12] was developed to specify a consistent interface between a variety of different state space representations (sparse-matrix, Kronecker, and matrix-diagram) and analytical solvers. The user creates a state space generator in the project which links with the existing libraries to create an executable that builds the state space and stores it in a local file. Then, the user can choose from a variety of analytical solvers, including the transient solver, the iterative steady state solver, the accumulated reward solver, the adaptive transient solver, the direct steady state solver, or the deterministic iterative steady state solver. The chosen solver reads in the state space and solves the reward variables defined in the project.

3. Building Large Models

Building and solving a model in Möbius is a multistep hierarchical approach. Atomic models are first constructed, and then may be composed with other atomic and composed models to build a larger system model. In this section, we describe the atomic and composed modeling formalisms that have been implemented in Möbius, focusing on formalisms that have been added since the original paper describing Möbius appeared.

3.1. Constructing Atomic Models

A wide variety of modeling formalisms have been implemented in Möbius, including *buckets and balls*, a simple extension of continuous time Markov chains, which includes arbitrary cardinality and transition distributions; *PEPA*, a stochastic process algebra with extensions to assign rates to activities; SANs, a stochastic extension to Petri nets; *Fault Trees*, a simple combinatorial dependability modeling formalism; and *MODEST* [4], a formal methods language from the University of Twente. In each case, the model editor implements the model-level AFI described earlier, generating model code that can be composed with other models without requiring knowledge of the semantics of a particular formalism.

The most recent atomic formalism implemented in Möbius supports building models with fault trees. Fault trees have the operational state of the system at their roots, and failure events at their leaves. Internal nodes of the fault tree are gates that implement the standard logical operations, *and*, *or*, *xor*, *k-of-n*, and *priority and*. Leaf and root nodes from fault trees can be shared with state variables in other models, allowing composition to represent portions of the system that cannot be expressed through a fault tree.

Note that the large collection of atomic model formalisms can also help reduce the learning curve for modelers on a team. Rather than force everyone to use a single formalism, the Möbius modeling framework offers the flexibility of allowing team members to model their subsystems in whichever formalisms they prefer, by allowing them to use the existing diversity of formalisms or build new ones as necessary.

3.2. Composing Models

Users can construct composed models in Möbius from atomic models or other composed models by placing the atomic models together in a structured process, creating a hierarchical model of the complete system. Möbius implements three composed model formalisms, all of which implement either *state sharing* or *action synchronization*. State sharing is accomplished by defining linked sets of variables, such as the places of a SAN, through *equivalence sharing*, meaning that both models have the same relationship to the shared state variables. *Replicate-Join* which existed in the initial release of Möbius, is a formalism that allows the definition of a composed model in the form of a tree whose leaf nodes are previously defined atomic or composed models, and whose non-leaf nodes are either a *Join* or *Replicate* node. *Join* nodes compose child submodels by state sharing. *Graph Composition* allows the construction of a composed model in the structure of a graph. Arcs linking models within the graph indicate a state sharing relationship. Both *Replicate-Join* and *Graph* composition techniques can utilize lumping techniques to automatically detect symmetries present in the structure of the model.

More recently, Möbius has added the ability to compose models using *action synchronization*. Larger models are formed by sharing actions between submodels instead of states. When two or more actions are synchronized they are replaced by a single representative action, whose enabling conditions are the result of the union of the enabling conditions of all composed sub-actions, and whose result is the union of the results of all composed sub-actions. The Möbius tool implements this composed model formalism by allowing the user to construct composed models using a tree whose leaf nodes are predefined atomic or composed models, and whose inner nodes are join nodes that synchronize on actions. To compose models in this fashion, the user selects the actions to synchronize, and defines the new time distribution function for the shared action. The new shared action is automatically set to have the correct enabling conditions and firing result.

All three methods allow breaking larger, more complex systems into smaller subsystems that can be easier to verify and understand. In addition this approach provides flexibility in model definition. In Section 6, we discuss the storage subsystem for a supercomputing system that was modeled using Möbius [14]. In that example, by defining the disks as an atomic submodel, it became simple to increase the scale of the modeled system by modifying a single parameter in the *Rep* node of the composed model. Careful design of the subsystems and their composition can allow a user to easily investigate many configurations of large-scale systems in this fashion.

4. Solving Large Models

Two difficult issues appear when a user is attempting to *solve* large models. First and foremost is the state space explosion problem [22], which occurs when the set of possible states of the represented system grows exponentially as more details of the system are added. A related problem is design space explosion [15], where the number of design configurations that the modeler intends to evaluate increases exponentially as the number of parameters or parameter values used to vary the model configuration increases.

Möbius addresses these issues by providing extensions that allow for efficient representation of the models being built and for efficient solution of performance variables defined on those models. Furthermore, the Möbius framework is built to allow integration of newer, state-of-the-art techniques as they are invented.

4.1. Efficient Representation

Möbius supports a rich variety of techniques to handle the state-space explosion problem. Models constructed based on sharing of state variables in Möbius's Replicate/Join or graph compositional framework expose structural symmetries that facilitate *lumping*. Möbius supports

state-level lumping, *model-level lumping*, and *compositional lumping* [10, 11, 20]. State-level lumping uses algorithms that can identify sets of states that are equivalent to one another in the sense that they can be replaced by a single state while preserving the Markovian property of the desired reward measures defined on the underlying CTMC. In the case of model-level lumping, symmetry detection algorithms are applied to the components of the composed model, using techniques that exploit formalism-dependent symmetries. In general, model-level lumping will not obtain as optimal a lumping as state-level lumping would. Compositional lumping takes the approach of state-level lumping, but applies it to the components of a composed model, and lumped equivalent components are formed. Furthermore, new algorithms allow symbolic representations of these lumped state spaces that further reduce their memory footprint by converting the new lumped representations into compact representations using MDDs. [11, 20] showed that these lumping techniques reduced state spaces by 1 to 2 orders of magnitude, enabling Möbius to solve systems with millions of states efficiently.

Möbius' state-level AFI [12] has been key to the integration of lumping and representation techniques. The AFI allows solution methods to communicate with models in Möbius . This approach separates state-space and state-transition-rate-matrix generation and representation from numerical solution techniques. The state-level AFI exposes to solvers the details of a model as a labeled transition system. The abstraction provided by the AFI allows the Möbius tool to separate the problem of efficent representation of the model from the problem of model solution. Solvers must implement access to a model through the Möbius state-level AFI, and can operate on a number of representations without any changes to the implementation of the solver. That enables existing numerical solution techniques to automatically work with the newer symbolic representations of CTMCs without any changes.

4.2. Simultaneous Solution of Alternative Design Configurations

One important use of simulation lies in comparing and choosing the best among different simulation models that represent competing or alternative designs before the actual deployment and implementation. In addition to the traditional discrete-event simulator, Möbius includes a new approach, which provides a methodology that exploits the structural similarity among alternative configurations and results in an efficient simulation algorithm that evaluates alternative configurations of a system simultaneously [15, 13]. In order to incorporate this new feature, it was necessary to make minor modifications to the state representation of the Möbius model-level AFI. The states in the models were upgraded to represent arrays of states (of different design configurations). With that update, simultaneous simulation showed that a significant speed-up can be achieved by the amortization of the event-set management required to change the state of the system through actions. Specifically, [15] showed that one can obtain 1 to 2 orders of magnitude speed-up in the evaluation of availability of a database server.

5. Analysis of Results

Analysis of results as a feedback process in building and evaluating systems is an integral part of modeling. The analysis enables modelers to update models, determine errors in the process of interpreting the real system, corroborate evidence of accuracy and correctness of model results to make sound judgments, and perform sensitivity analysis on model parameters.

Möbius provides extensive support for the analysis of results, which are either integrated into the Möbius framework or as external tools. Möbius has integrated database support to add results from solutions generated from experiments solved using numerical or simulation techniques into an external SQL database. The result include the model parameters, experiment parameters such as batch sizes, and time of execution, and other related information. Möbius enables users to graph the experimental results inside the tool itself. This helps Möbius users tweak their parameters or models if necessary. In addition, the Möbius tool suite provides an external *DataManager* tool that enables users to perform richer analysis of the experimentally generated data. By reviewing changes in the history of reward measures and their values, users can sequence and visualize changes and variations in their Möbius system models as they update the models over time.

While Möbius allows models to be evaluated using numerical or simulation techniques, it also provides an intuitive mechanism to visualize the working of models in action. Möbius enables users to generate traces that can be fed to tools like Traviando to perform complex analysis of system behavior [19]. Traviando is an event browser that helps a modeler select sequences of events of interest by filtering unnecessary information out of the Möbius traces.

If one needs to perform analysis efficiently, the Möbius tool also supports Design of Experiments, with which the experiment sets can be produced in a controlled manner that provides sensitivity analysis of the model's parameter values [29]. Users can generate and visualize the interaction between results and varying parameter values using plots and graphs, using metrics such as autocorrelation and regression coefficients, like residual errors.

6. Use Cases

Möbius has been used in a broad range of disciplines to analyze and evaluate discrete-event systems. Over 360 licenses (site or individual) have been issued for its use.

Due to its extensible and flexible framework the Möbius tool has been applied to various fields, from probabilistic security analysis [28, 27] to dependability analysis of low-earth-orbit satellites [1]. In this section, we provide a brief overview of certain real-world applications and problems for which Möbius was used as the modeling and evaluation tool.

Validation of a Publish-Subscribe System The validation of the survivability of a complex publish-subscribe system using heterogeneous components, from hardware to software used for communication, is quite challenging. Singh et al. provide a systematic top-down approach that precisely defines the survivability of such a system in a hostile environment by decomposing the system into manageable components [26, 27]. The validation process used Möbius to conduct model-based experiments to measure the survivability in real deployment by testing the system under various hostile attack scenarios. The results and analysis were later used to fine-tune the actual design, implementation, and deployment of the real system.

ABE System The Möbius tool was used to develop a SAN model that uses failure rates computed from real logs to predict the reliability and availability of the storage architecture of the ABE cluster at the National Center for Supercomputing Applications [14]. The analysis of ABE provides insight into a new design approach that will enable system designers to integrate the trace-based analysis of parameter values from real system data-logs into modeling tools such as Möbius.

Quantifying Cell Phone Virus Propagation Van Ruitenbeek et al. collaborated with France Telecom to investigate the problem of quantifying virus propagation in mobile phones [28]. Virus infection and propagation models were built with Möbius, and parametrized to test a wide range of virus behavior. Van Ruitenbeek's team used the Möbius discrete-event simulator, and database to compare the effectiveness of mobile phone virus response mechanisms.

Molecular Biology Möbius has also been used outside of the domains of reliability, dependability, and performance. *UltraSAN*, the predecessor to Möbius , was used by Goss and Peccoud in [16, 17] to simulate molecular networks by encoding them as SANS. The authors cite the standardized format of atomic models in [17] as facilitating collaboration between researchers. The modeling framework allowed exploration of previously unknown rates in the molecular networks, and helped to provide an explanation for the maintenance of a protein found in the studied bacteria.

7. Related Work

We provide brief overviews of a few related tools. For a complete discussion of related work, please refer to [23].

One of the earliest tools to combine multiple modeling formalisms and solution formalisms into a single tool was SHARPE [18]. SHARPE allowed results to be exchanged between formalisms in the form of exponential-polynomial distribution functions. In Möbius, we support submodels to share state and actions (of any distribution type). Like Möbius, SHARPE provides several formalisms to represent systems. However, Möbius also provides an interface that enables solution techniques to be agnostic of the underlying state-space representation [12].

SMART [6] is another tool that enables multiple formalisms to interact by exchanging results, possibly repeatedly, using fixed-point iterations. SMART models are represented as SPNs and queuing networks. Also, like Möbius, SMART is implemented to allow for additional solvers to be implemented and integrated into the tool.

The DEDS (Discrete Event Dynamic System) [3] toolbox implements its multi-formalism approach by converting the specifications into a unified "abstract Petri net notation." From that formalism, DEDS provides several functional and quantitative analysis methods.

GreatSPN 2.0 is a software package for the modeling, validation, and performance evaluation of distributed systems using Generalized Stochastic Petri Nets and their colored extension: Stochastic Well-formed Nets [2]. Like Möbius, GreatSPN 2.0 can execute different analysis modules in a distributed computing environment [5]. New analysis modules can be added to GreatSPN 2.0, via an interface similar to state-level AFI.

While all of these tools provide some kind of extensible framework, Möbius provides a complete solution in which large systems represented by the model formalism, the underlying state space generated for the solver, and the solution techniques can truly be independent of one another, and each of these components can be upgraded with newer techniques in order to obtain results and perform analysis quickly.

8. Conclusion

System-level modeling of the dependability of practical information-technology-based systems is a challenging undertaking. The challenge comes from both the inherent complexity of such systems, and the need to represent many different aspects of a system's behavior in order to accurately predict its dependability, security, or performance. The Möbius modeling tool was created to facilitate the evaluation of such systems by implementing a wide variety of modeling formalisms and solution methods and supporting a flexible way to add new modeling formalisms and solution methods (through the model- and state-level AFIs). The evidence of extensibility of Möbius comes from the many new formalisms and solution methods that have been implemented, both by the Möbius team and by external researchers, since the original release. The evidence of its success in modeling IT-based systems comes from the large

number of practical studies that have been carried out, as illustrated in Section 6. While it is clear that the tool is quite useful in its current form, work continues with the aim of adding features that would enable its wide acceptability by simplifying its ease of use and interaction with the user.

9. Acknowledgements

We would like to acknowledge the contributions of the current and former members of the Möbius group as well as the contributions by our collegues at other institutions. We would also like to thank Jenny Applequist for her editorial comments.

References

[1] E. Athanasopoulou, P. Thakker, and W. H. Sanders. Evaluating the dependability of a LEO satellite network for scientific applications. In *QEST*, pages 95–104, 2005.

[2] S. Baarir, M. Beccuti, and G. Franceschinis. New solvers for asymmetric systems in GreatSPN. In *QEST*, pages 235–236, 2008.

[3] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of DEDS. In R. Puigjaner, N. N. Savino, and B. Serra, editors, *Proc. of the 10th Int. Conf. on Computer Perf. Eval.: Modeling Tech. and Tools*, pages 356–359, Palma de Mallorca, Spain, Sept. 1998.

[4] H. Bohnenkamp, T. Courtney, D. Daly, S. Derisavi, H. Hermanns, J. Katoen, R. Klaren, V. V. Lam, and W. Sanders. On integrating the Möbius and MODEST modeling tools. *DSN*, pages 671–671, June 2003.

[5] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: graphical editor and analyzer for timed and stochastic petri nets. *Perform. Eval.*, 24(1-2):47–68, 1995.

[6] G. Ciardo and A. Miner. Smart: Simulation and Markovian analyzer for reliability and timing. *Proc. of IEEE Int. Computer Perf. and Dependability Symp.*, pages 60–, Sept. 1996.

[7] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The Möbius modeling tool. In *Proc. of the 9th Int. Workshop on Petri Nets and Perf. Models*, pages 241–250, Sept. 2001.

[8] D. Deavours and W. H. Sanders. Möbius: Framework and atomic models. In *10th International Workshop on Petri Nets and Performance Models*, pages 251–260, Sept. 2001.

[9] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Möbius framework and its implementation. *IEEE Transactions on Software Engineering*, 28(10):956–969, Oct. 2002.

[10] S. Derisavi, P. Kemper, and W. H. Sanders. Symbolic state-space exploration and numerical analysis of state-sharing composed models. *Linear Alg. and its App.*, 386:137–166, Jul. 2004.

[11] S. Derisavi, P. Kemper, and W. H. Sanders. Lumping matrix diagram representations of Markov models. In *DSN*, pages 742–751, Jun. 2005.

[12] S. Derisavi, P. Kemper, W. H. Sanders, and T. Courtney. The Möbius state-level abstract functional interface. *Perform. Eval.*, 54(2):105–128, 2003.

[13] S. Gaonkar, T. Courtney, and W. H. Sanders. Efficient state management to speed up simultaneous simulation of alternate system configurations. In *Proc. of the Int. Med. Modelling Multiconference*, pages 31–36, 2006.

[14] S. Gaonkar, E. Rozier, A. Tong, and W. H. Sanders. Scaling file systems to support petascale clusters: A dependability analysis to support informed design choices. In *DSN*, pages 386–391, 2008.

[15] S. Gaonkar and W. H. Sanders. Simultaneous simulation of alternative system configurations. In *PRDC*, pages 41–48, Changsha, Hunan, China, 2005.

[16] P. J. E. Goss and J. Peccoud. Quantitative modeling of stochastic systems in molecular biology using stochastic petri nets. In *Proc. Natl. Acad. Sci. U.S.A.*, volume 95, pages 6750–6755, 1998.

[17] P. J. E. Goss and J. Peccoud. Analysis of the stabilizing effect of Rom on the genetic network controlling plasmid ColE1 replication. In *Proc. Pacific Symp. for Biocomputing*, pages 65–76, 1999.

[18] J. Horch. A sharp modeling approach. *Software, IEEE*, 15(4):88–89, Jul/Aug 1998.

[19] P. Kemper and C. Tepper. Automated analysis of simulation traces - separating progress from repetitive behavior. In *QEST*, pages 101–110, 2007.

[20] M. G. McQuinn, P. Kemper, and W. H. Sanders. Dependability analysis with Markov chains: How symmetries improve symbolic computations. In *QEST*, pages 151–160, 2007.

[21] Möbius Team. *The Möbius Manual, www.mobius.illinois.edu*. University of Illinois, Urbana Champaign, Urbana, IL – 61801, 2008.

[22] D. M. Nicol, W. H. Sanders, and K. S. Trivedi. Model-based evaluation: From dependability to security. *IEEE Trans. on Dependable and Secure Comp.*, 1(1):48–65, 2004.

[23] W. H. Sanders. Integrated frameworks for multi-level and multi-formalism modeling. In *Proc. of 8th International Workshop on Petri Nets and Performance Models*, pages 2–9, Sept. 1999.

[24] W. H. Sanders and J. F. Meyer. Stochastic activity networks: Formal definitions and concepts. In E. Brinksma, H. Hermanns, and J. P. Katoen, editors, *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 315–343, Berg en Dal, The Netherlands, 2001. Springer.

[25] W. H. Sanders, W. D. Obal, M. A. Quereshi, and F. K. Widjanarko. The UltraSAN modeling environment. *Perf. Eval.*, 24(1):89–115, Oct.-Nov. 1995.

[26] S. Singh, A. Agbaria, F. Stevens, T. Courtney, J. F. Meyer, W. H. Sanders, and P. Pal. Validation of a survivable publish-subscribe system. *Int. Sci. Journal of Comp.*, 4(2), 2005.

[27] F. Stevens, T. Courtney, S. Singh, A. Agbaria, J. Meyer, W. Sanders, and P. Pal. Model-based validation of an intrusion-tolerant information system. *SRDS*, pages 184–194, Oct. 2004.

[28] E. Van Ruitenbeek, T. Courtney, W. H. Sanders, and F. Stevens. Quantifying the effectiveness of mobile phone virus response mechanisms. *DSN*, pages 790–800, 2007.

[29] P. G. Webster. Design of experiments in the Möbius modeling framework. Master's thesis, U. of Illinois at Urbana-Champaign, 2002.