The Möbius Modeling Environment: Recent Extensions - 2005*

Tod Courtney, Salem Derisavi, Shravan Gaonkar, Mark Griffith,
Vinh Lam, Michael McQuinn, Eric Rozier, and William H. Sanders
Department of Electrical and Computer Engineering and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL, U.S.A.
{tod,derisavi,gaonkar,mgriffth,lam,mcquinn,ewdr,whs}@crhc.uiuc.edu
http://www.mobius.uiuc.edu

Abstract

The Möbius modeling environment is an extensible framework for discrete-event system analysis that allows multiple formalisms and solution techniques to easily interoperate, and new modules to be easily added. The basis of the framework is an abstract functional interface that defines the behavior and data to be shared among modules. New formalism and solver modules continue to be added to the tool. This paper describes recent additions to Möbius, including a fault tree model definition formalism, a model composition formalism based on action synchronization, improvements in reward model definition, and additional lumping capabilities in the symbolic state space generator.

1 Möbius Tool

Möbius is a discrete-event system analysis tool that has been widely used for the analysis of dependability and performability properties of systems. Recently, Möbius has also been applied to the analysis of system survivability and assurance [1], [2]. Möbius supports multiple modeling formalisms as well as multiple solution techniques. It was created to satisfy the need for a general, common modeling environment to support the development and comparison of new modeling technologies, and avoid the need to continually reinvent and reintegrate older, established technologies. The Möbius framework consists of an abstract functional interface (AFI) [3] that makes it possible for modules to interact with each other in a unified manner.

The AFI is implemented by a set of C++ base classes that represent common components of discrete-event models: state variables, actions, and models. *State variables* hold the state of the model, *actions* generate events that change the state of the model, and *models* represent a component in a system and contain sets of actions and state variables.

This material is based upon work supported by a gift from Pioneer Hi-Bred International, Inc., and Grant No 0086096 from the National Science Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Pioneer Hi-bred or the National Science Foundation.

The Möbius tool architecture is designed to be flexible and portable. The front-end of the architecture consists of a Java-based graphical user interface. The back-end is written in C++ to attain efficient execution of the models and high performance for the solution techniques. Modules can either implement model definition formalisms or provide new solution algorithms. New modules can be easily incorporated into the tool to expand the set of available functionalities. Further details on the Möbius framework and tool can be obtained from `www.mobius.uiuc.edu`.

2 New Capabilities

Several new features have been included in releases of the Möbius tool during the past year. These features include two new model definition formalisms, unified support for reward variables across all solution techniques, support for the evaluation of reward variables at multiple time points, and expanded lumping capabilities within the symbolic state space generator.

2.1 Model Definition Formalisms

Simple systems can often be represented by a single model constructed with a single modeling formalism. For more complex systems, it is often convenient to use hierarchical representations of the system, made by decomposing the system into components, constructing a separate model for all of the components in the system, and then combining the component models into the model of the system. To support the representation of both simple and complex systems, model definition formalisms within Möbius fall into two general categories: formalisms for the specification of basic model behavior, and formalisms that support the composition of models into larger, more complex models.

Fault Trees We have added to Möbius a new formalism that supports model definition using fault trees. Fault trees are an effective way to represent the relationship between sources of failure within a system. The top-level node of the tree represents the operational state of the system. The leaf nodes represent failure events that could occur in the system. The failure events are connected to the leaf node

using gates that represent standard logical operations, such as *and*, *or*, *xor*, *k-of-n*, and *priority and*.

Fault trees can be incorporated into composed models through sharing of the leaf and root nodes with state variables from other models. Using model composition, it is possible to create hierarchies of fault trees. Also, additional aspects of the system that cannot be captured within the fault tree itself, such as repair operations, can be incorporated into the system model through composition of the fault tree model and models constructed in other formalisms.

Action Synchronization Composition Action synchronization is a common composition technique that allows larger models to be formed from component models by synchronizing actions found within each component model. In effect, when two actions are synchronized, they are replaced by a single representative action that is enabled in states in which both of the original actions are enabled. When fired, the representative action performs the behavior of both of the original actions. In the implementation of action synchronization in Möbius, the distribution function of the new synchronizing action is specified directly by the modeler when the synchronization is defined, allowing for a large degree of flexibility of behavior. Action synchronization complements the two forms of state-sharing composition, Rep-Join trees and generalized Graph composition, that were available in previous versions of Möbius.

2.2 Reward Model Definition Improvements

In Möbius, reward models are used to define reward variables, which are measures of interest that are evaluated and reported when the system is analyzed. Two significant improvements have been made to the way reward models are defined. First, individual reward variables can be defined for multiple time points (or multiple intervals of time), rather than individual times as in previous versions. Second, reward variables are handled in a consistent manner by all solvers. Solvers now recognize the type of variable (instant of time, interval of time, or steady-state) defined by the reward model, and only solve variables that correspond to type that they support. Also, all transient analytical solution techniques reference the definition of measurement time as defined in the reward model, instead of requiring the modeler to re-specify the time points as solver parameters.

2.3 Symbolic State Space Generation

We have integrated our compositional lumping algorithm [4] with the existing "model-level" lumping techniques [5] in Möbius. In [5], we extended previous work on Multi-valued Decision Diagrams (MDDs) and Matrix Diagrams (MDs) [6] to composed models that share state variables. The extension combines Replicate/Join-based lumping techniques, which are based on the equivalence of replicas in a Replicate node, with largeness-tolerance techniques that use MDDs and MDs. Our efforts have resulted in a new algorithm that symbolically generates the lumped state space and the state transition rate matrix of a hierarchical Replicate/Join model in the form of an MDD and an MD data structure, respectively.

In [4], we designed the first efficient compositional lumping algorithm for exact and ordinary lumping of Markov chains represented as MDs. One advantage of our compositional lumping algorithm over other formalism-specific ones is that our formulation is simple and easy to understand. More importantly, it is applicable to an MD regardless of the formalism of the model from which the MD was generated. More specifically, it works for any model formalism for which there is a state-space generation algorithm that generates an MD or Kronecker* representation of the underlying CTMC's state-transition rate matrix. The impact of the work is apparent from the observation that many authors have recently used MDs to represent very large matrices of Markovian models specified in a variety of modeling formalisms.

3 Acknowledgments

We would like to acknowledge the contributions to the action-synchronization-based composition by Kai Lampka and Markus Siegle from the Universität der Bundeswehr München. We would also like to acknowledge the previous contributions of the former members of the Möbius group (A. Christensen, G. Clark, D. Daly, D. Deavours, J. Doyle, G. Kavanaugh, J. Sowder, A. Stillman, P. Webster, and A. Williamson) as well as the contributions by P. Buchholz, P. Kemper, and C. Tepper from the Universität Dortmund; H. Hermanns from the Universität des Saarlandes; and H. Bohnenkamp, B. Haverkort, D. Janecek, J. Katoen, and R. Klaren from the Universiteit Twente. Finally, we would like to thank Jenny Applequist for her editorial comments.

References

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

[2] 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 Computing*, to appear.

[3] 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 Trans. on Soft. Eng.*, vol. 28, no. 10, pp. 956–969, Oct. 2002.

[4] S. Derisavi, P. Kemper, and W. H. Sanders, "Lumping matrix diagram representations of Markov models," in *Proc. DSN 2005*, Jun./Jul. 2005, to appear.

[5] ——, "Symbolic state-space exploration and numerical analysis of state-sharing composed models," *Linear Alg. and its App.*, vol. 386, pp. 137–166, Jul. 2004.

[6] G. Ciardo and A. Miner, "A data structure for the efficient Kronecker solution of GSPNs," in *Proc. 8th Int. Workshop Petri Nets and Perf. Models*, 1999, pp. 22–31.

*Any Kronecker expression can also be represented as an MD.