The Möbius Modeling Environment: Recent Developments*

Tod Courtney, David Daly, Salem Derisavi, Shravan Gaonkar,
Mark Griffith, Vinh Lam, 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, ddaly, derisavi, gaonkar, mgriffth, lam, whs}@crhc.uiuc.edu
http://www.crhc.uiuc.edu/PERFORM

Abstract

The Möbius modeling tool provides an infrastructure to support multiple interacting formalisms and solvers, and is extensible in that new formalisms and solvers can be added to the tool in such a way that they can interact with those already implemented without requiring additional changes to the previously implemented ones. We have continued to add features to the Möbius tool in order to enhance the extensibility of the tool and to allow increased interaction between new and different formalisms and solvers. In this paper, we discuss some recent additions to Möbius, including expanded state variable types, a fixed-point mechanism called "connection," and improvements to CTMC solution that includes a new state level abstract functional interface.

1 Möbius Tool

Möbius is a system-level performance and dependability modeling tool that was motivated by the observations that no one formalism is best for building and solving models, that no single solution method is appropriate for solving all models, and that new formalisms and solution techniques are often hindered by the need to build a complete tool to handle them. We dealt with these three issues by defining a broad framework (a formal, mathematical specification of model construction and execution [1]) in which new modeling formalisms and model solution methods can be easily integrated. In implementing the framework we defined an *abstract functional interface* (AFI) [2], which is realized as a set of functions that facilitates inter-model communication as well as communication between models and solvers.

The Möbius tool architecture is separated into two different logical layers: model specification and model execution. Model specification in the tool is done through graphical user interfaces in Java, while all model execution is done exclusively in C++ to attain efficiency and high performance. The Möbius framework provides a very general approach to specifying a model in a particular formalism. Every formalism has a separate editor for specifying a particular piece of the model. The C++ files produced by the editor are compiled, and the tool links the object code with formalism libraries and solver-specific libraries, giving users the flexibility necessary to solve systems in different domains and representations.

Further details on the Möbius tool and its existing functionality can be obtained from www.mobius.uiuc.edu.

This material is based upon work supported by the National Science Foundation under Grant Nos. 9975019 and 0086096. 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 National Science Foundation.

2 New Features

The Möbius tool is constantly being updated, and new versions are released every six months with bug fixes and new features. In this section, we describe features that were added to the tool during the last year.

2.1 Basic Functionality

Recent work done in collaboration with researchers from the University of Twente has expanded the state variable capabilities of Möbius to support all of the basic datatypes as well as some complex data types, such as structures and arrays. These expanded capabilities allow composed models to define complex state-sharing relationships, such as sharing of fields of a structure with elements of an array. Reward variables can also be defined on the same structure and array data types. The new expanded framework enables other formalisms to incorporate complex data types through the AFI into their formalisms.

Additionally, support has been added for the efficient state space expansion of phase type delay distributions. As part of this extension, models created in the SAN formalism can be solved analytically even if they have activities with Erlang delay distributions.

2.2 Generic Connection Formalism

Connection is a type of model formalism that allows multiple solvable models to be iteratively solved and their results to be exchanged on each iteration. Each solvable model is solved using a collection of results from other solvable models. The results from other solvable models

are accessed through the Möbius results database [3], and those results are used to set global variables.

We have built a general infrastructure for the implementation of different connection formalisms we used four basic components: *solvable models*, *conduits*, *connection functions*, and *database accesses*. The solvable models are the set of models to solve, and the conduits specify how results are exchanged between the models. Connection functions allow an arbitrary transformation of the results, and the database access allows the user to specify arbitrary access to data in the Möbius database, in addition to the automated access supported by the solvable models and connection functions.

We have implemented a generic connection formalism using the connection infrastructure. The user can specify the iteration order of the nodes in the model, a stopping criterion, and a minimum and maximum number of iterations to perform for the fixed-point solution. Additionally, the modeler may specify a subset of the models that should be solved only once, with the rest of the models being solved in every iteration.

The generic connection formalism can also be used as a basis for building specialized or more elaborate connection formalisms. For instance, it would be easy to extend the formalism to support the fixed point solution of queuing networks, with a specific set of results to pass and specialized connection functions to combine and split the output of each queue.

2.3 CTMC Analysis

State-Level AFI The rich variety of techniques to deal with the state-space explosion problem, and the fact that many numerical solution methods share similar basic operations, have motivated us to develop a state-level, as opposed to the initial model-level, AFI for Möbius. By using the state-level AFI [4], we separate state-space and state-transition-rate-matrix generation and representation issues from issues related to the solution of the resulting state-level models. This means that we can create and implement numerical solution methods that do not require information about the data structures of the state space and the transition rate matrix. The Möbius state-level AFI has advantages for both tool developers and tool users. Most importantly, our approach avoids redundant reimplementations of the three steps (model specification, state-space and state-transition-rate-matrix generation, and numerical analysis) taken when solving models numerically using state-based methods.

Symbolic Representation In [5, 6], we have extended previous work on Multi-valued Decision Diagrams (MDDs) and Matrix Diagrams (MDs) [7] 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. In particular, 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. Moreover, by designing a variant of the MDD data structure, we have been able to efficiently enumerate the entries of the state transition rate matrix, thus making solutions of large CTMCs efficient.

3 Acknowledgments

We would like to acknowledge Henrik Bohnenkamp, Ric Klaren, Johan Gorter, Holger Hermanns, and Joost-Pieter Katoen from the University of Twente for their work on integrating MODEST with Möbius. We would also like to recognize Holger Hermanns for his work on optimal lumping, Peter Kemper for his work on the state-level AFI, and Kai Lampka for his work in implementing action synchronization-based composition. We would also like to acknowledge the work done by former members of the Möbius group: Amy L. Christensen, Graham Clark, Daniel D. Deavours, Jay M. Doyle, G. P. Kavanaugh, John M. Sowder, Aaron J. Stillman, Patrick G. Webster, and Alex L. Williamson. Finally, we would like to thank Jenny Applequist for her support of the PERFORM modeling group, in general, and the editing of this paper, in particular.

References

[1] 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, October 2002.

[2] J. M. Doyle, "Abstract model specification using the Möbius modeling tool," Master's thesis, University of Illinois at Urbana Champaign, January 2000.

[3] A. L. Christensen, "Result specification and model connection in the Möbius modeling framework," Master's thesis, Univ. of Illinois at Urbana-Champaign, 2000.

[4] S. Derisavi, P. Kemper, W. H. Sanders, and T. Courtney, "The Möbius state-level abstract functional interface," *Perf. Eval.*, vol. 54, no. 2, pp. 105–128, Oct. 2003.

[5] S. Derisavi, P. Kemper, and W. H. Sanders, "Symbolic state-space exploration and numerical analysis of state-sharing composed models," *Linear Algebra and its Applications*, vol. 386, pp. 137–166, July 2004.

[6] ——, "Symbolic state-space exploration and numerical analysis of state-sharing composed models," in *Proc. of Numerical Solution of Markov Chains (NSMC)*, 2003, pp. 167–189.

[7] G. Ciardo and A. Miner, "A data structure for the efficient Kronecker solution of GSPNs," in *Proceedings of the 8th Int. Workshop Petri Nets and Performance Models*. IEEE CS Press, 1999, pp. 22–31.