The Möbius Modeling Environment

Tod Courtney, David Daly, Salem Derisavi, 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, lam, whs}@crhc.uiuc.edu
http://www.crhc.uiuc.edu/PERFORM

***Abstract*— Despite the development of many modeling formalisms and model solution methods, most tool implementations support only a single formalism. Furthermore, models expressed in a chosen formalism cannot be combined with models expressed in other formalisms. This paper describes a modeling tool called Möbius, which 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 such that they can interact with those already implemented without requiring additional changes to the previously implemented ones. We discuss the initial implementation of Möbius, and the recent addition of different formalisms and solution techniques to the tool.**

I. Introduction

Möbius is a system-level performance and dependability modeling tool. The motivations for building the Möbius tool were the observations that no 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 define 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. This AFI also allows the modeler to specify different parts of the model in different formalisms.

In the rest of the paper, we describe the Möbius tool and its features. We describe the tool as it was originally released, and the new formalisms and features added since the original release. For more on the theory and implementation of Möbius, please see [3]. In section II we describe the basic Möbius framework, before discussing the framework's implementation and initial features in section III. In section IV we review the new features in Möbius, including several new formalisms that demonstrate the flexibility of Möbius.

II. Möbius Framework

We begin with a brief overview of the concepts of a formalism and a model in the Möbius framework. The Möbius framework provides a very general way to specify a model in a particular formalism. We define a *formalism* as a language for expressing a model within the Möbius framework, frequently using only a subset of the options available within the framework.

We define models within the Möbius framework using a few basic concepts. A *model* is a collection of state variables, actions, and reward variables expressed in some formalism. Briefly, *state variables* hold the state information of the model. *Actions* change the state of the model over time. *Reward variables* are ways of measuring something of interest about the model.

Although the basic elements of a model are very general and powerful, formalisms need not make use of all the generality. In fact, it may be useful to restrict the generality in order to exploit some property for efficiency. The purpose of some formalisms is to expose these properties easily, and to take advantage of them for efficient solution. Möbius was designed with this in mind.

In order to improve the reusability of models already built, it is useful to classify models as follows. The most basic category is that of "atomic models." An *atomic model* is a self-contained (but not necessarily complete) model that is expressed in a single formalism. Several models may be structurally joined together to form a single larger model, which is called a *composed model*. A composed model is itself a model, and may itself be a component of a larger composed model. A model that is more loosely connected by the sharing of solutions is called a *connected model*. Next, we describe how we implement this framework as a tool.

III. Möbius Tool

The first step in implementing the Möbius framework was to define the AFI that is at the core of the tool. We have implemented the AFI as a set of C++ base classes from which all models must be derived. In doing so, we defined the functional interfaces as pure virtual methods. This requires that any formalism implementor define the operation of all the methods in the functional interface. In the same fashion, we constructed C++ base classes for other Möbius framework components, including actions, state variables, and reward variables.

The Möbius tool architecture (see Figure 1) is separated into two different logical layers: model specification and model execution. All model specification in our tool is done through Java graphical user interfaces, and all model execution is done exclusively in C++. We decided to implement the executable

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.

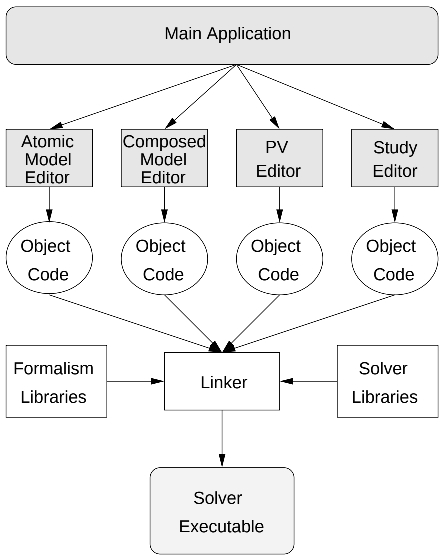

Fig. 1. Möbius Architecture.

models in C++ for performance reasons. Every formalism has a separate editor for specifying a particular piece of the model. Editors produce compilable C++ code as output so that the final executable model is specified entirely within C++. The C++ files produced by the editor are compiled, and the tool links the object code with formalism libraries and solver-specific libraries.

The Möbius tool originally supported the following formalisms and capabilities:

SAN The stochastic activity network atomic formalism [4].

Replicate-Join A composition formalism that composes models using two constructs: replicate and join [5].

Graph Composer A composition formalism that allows the construction of a composed model as an arbitrary graph of submodels with shared state [6].

Rate-Impulse Reward A formalism for the specification of reward variables whose values are determined by a set of state-based rates and impulse functions [7].

Studies Throughout all phases of model specification, global variables can be used as input parameters. These editors allow the modeler to specify the values of those global variables.

Discrete-Event Simulator This generic simulator allows any model to be simulated for transient or steady-state reward measures. It also allows the simulation to be distributed across a heterogeneous set of workstations, resulting in a near-linear speed-up.

State-Space Generator This module creates a Markov process description for a model that has exponentially distributed delays between state changes. The output of the state-space generator is used as an input for many different analytical solvers.

Analytical Solvers Several analytical solvers are implemented in the Möbius tool. They include both transient and steady-state solvers.

IV. New Formalisms and Features

In the process of developing the first Java interfaces, we constructed several Java class packages that facilitate the construction of graphical user interfaces for the Möbius tool. We have generally exploited these utilities in adding new formalisms and functionality to the tool, and they have made the additions much easier. We roughly divide the new features into the following six categories: basic functionality, atomic model formalisms, parameterization and results, state-based techniques, and other expanded functionalities. We discuss these six categories in order.

A. Basic Functionality

Recent work done in collaboration with researchers from the University of Twente have expanded the state variable capabilities of Möbius to support all basic types of state data and also the complex data types of structures and arrays. These expanded capabilities allow composed models to define complex state-sharing relationships, such as sharing fields of a structure with elements of an array. Reward variables can also be defined on the same structure and array data types.

Other work has been done on the composition of models that synchronize on actions. Action synchronization naturally occurs in many modeling formalisms, and can be used to create efficient state-space representations.

B. Atomic Model Formalisms

The AFI makes it possible for new formalisms to be added to the Möbius framework easily and quickly. The four atomic formalisms described below have all been added to Möbius, and show the flexibility of the interface and the diversity of formalisms it can accommodate. It should be emphasized that models built using these formalisms interact seamlessly through the AFI among themselves, with the solvers or simulator, or with other components in the Möbius tool.

Buckets and Balls (B&B [8]) is a fairly simple formalism that defines a number of extensions over Continuous Time Markov Chains (CTMCs) and makes possible the expression of more complex model behavior than CTMCs could. Among these extensions are arbitrary cardinality and distribution function for the transitions of a CTMC. Any B&B model can be converted to an equivalent SAN model. However, we believe that the B&B formalism enables us to easily and quickly design *simple* models for which the equivalent SAN model would have more components and would therefore be more difficult to understand.

Performance Evaluation Process Algebra (PEPA) [9] is a stochastic process algebra that extends classical process algebra with the capability to assign rates to activities. It is an algebraic language intended for modeling distributed systems. Several attractive features of PEPA are compositionality of submodels, formality of the language that is grounded in an algebraic system, and abstraction of details in submodels for building complex models. Modeling with PEPA is similar to constructing a system of algebraic equations.

MODEST [10], [11] is a stochastic process algebra language. The language features include constructs such as exception handling, modularization, data structures, and structuring mechanisms imposed via composition and abstraction. Semantically, the language supports nondeterministic and probabilistic branching and timing. As a whole, the language resembles modern programming languages, so it is intuitive to use and permits formal reasoning about processes being modeled. MODEST and PEPA are quite different from other formalisms, such as SAN and B&B, in that they are text-based (as opposed to graphics-based) and that the concept of processes is central to constructing models with them.

The three atomic formalisms addressed in this section represent integrated formalisms within the Möbius tool. While it often is appropriate to fully integrate formalisms into Möbius, there are cases when less integration is desired, such as an integration between Möbius and another fully functional and developed tool. To support a less coupled integration, we have developed a new *external formalism interface*. With this interface, the external tool does the work of converting the formalism to Möbius AFI-based classes and compiling those classes into a library. The external tool then generates a text file containing information describing the model, such as names of state variables, actions, and the compiled C++ library. With this information, Möbius can use the externally defined atomic model within a composed model or reward definition as if it had been defined within Möbius.

C. Parameterization and Results

As noted previously, *global variables* can be used to parameterize model characteristics. Models are solved after each global variable is assigned a specific value. Each specific assignment forms an *experiment*. When an experiment is solved, its solution consists of several results. There are three new features related to global variables and results in Möbius: database support, a design-of-experiment editor, and a generic connection editor.

1) Database support and viewer: Möbius has added support to put all results from the solution of an experiment into an external SQL database [12]. It stores all of the results, global variables, and other related data (e.g., number of batches or cpu time). The database can be accessed by formalisms within the Möbius tool, and can be queried through a command-line-based browser that enables advanced queries of the data.

The Möbius database has been implemented in a layered manner in order to support multiple database software packages. The lowest level is the virtual database layer, which contains all interaction with the external database. Möbius currently supports the PostgreSQL database, and it should be possible to add support for different databases by making minimal changes.

2) Design-of-experiment editor: The Möbius tool supports several study editors, the most sophisticated of which is based on a Design of Experiments approach (DOE) [13]. A DOE study generates a set of experiments and then analyzes the reward variable solutions to determine how the chosen global variables affect the reward variables. The results are gathered from the results database. Sensitivity analysis can measure the effects of all global variables and their interactions on each solved reward variable. In addition, the global variable values that produce optimal reward variable values can be determined.

3) Generic connection editor: Additionally, we have added support for connection editors. *Connection* is a type of model formalism that allows multiple solvable models to be solved and their results to be exchanged. Each solvable model is solved with a collection of results from other solvable models. The results from other solvable models are accessed through the results database, and those results are used to set global variables. The models can then be solved using a standard fixed-point iteration.

We have built a general infrastructure for the implementation of different connection formalisms, using four basic components: *solvable models*, *conduits*, *connection functions*, and *database accesses*. The solvable models are just 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 database, in addition to the automated access supported by the solvable models and connection functions. It is easy to create connection formalisms by specifying the order in which models should be solved and a stopping criterion, as well as any additional features that are desired.

D. State-Based Techniques

Once models are specified and parameterized, they can be solved either analytically or through simulation. We have added and improved several capabilities for state-based analysis.

1) 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 [14], 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. In particular, our approach, when used together with the Möbius model-level AFI, 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. That significantly reduces the effort that is necessary to implement, validate, and evaluate new approaches. Furthermore, it allows users to perform direct comparison of alternative approaches, without having to reimplement the work of other researchers; thus, they avoid the risk of being unfair when doing a comparison. Finally, it facilitates cooperation among researchers in developing new solution methods and combining existing ones. In short, we achieve a situation in which research results that focus on model reduction, state-space exploration, state-transition-rate-matrix representation, or analysis can be developed independently but be used with one another.

2) Optimal Lumping: In [15], we improved upon existing algorithms for constructing the optimal lumping quotient of a finite Markov chain and presented one with a time complexity of $\mathcal{O}(m \lg n)$, where n and m are, respectively, the number of states and transitions of the Markov chain. This algorithm has been implemented and will be integrated in the tool in the near future.

3) Symbolic Representation: Currently, we are working on extending previous work on Multi-valued Decision Diagrams (MDDs) [16] and Matrix Diagrams (MDs) [17] to composed models that share state variables. The extension combines Replicate/Join-based lumping techniques, which have been applied to state-sharing composed models, with largeness tolerance techniques that use MDDs and MDs. In particular, our efforts have resulted in a new algorithm that symbolically generates the state space and the state transition rate matrix of a hierarchical model (which is built using join and replicate operators [5]) in the form of an MDD and an MD data structure, respectively.

E. Other expanded functionality

In addition to incorporating new formalisms and solution techniques into Möbius, we continue to develop the tool itself to expand its functionality and improve its usability. The improvements include changes to the simulator, the studies, and the project, as well as other assorted improvements. The networking and I/O routines of the simulator have been redesigned to support distributed simulation on both Unix and Windows workstations, to allow the definitions of machine clusters, and to allow export of the simulation observation data for offline processing. The Study editor has been improved to allow easier enabling and disabling of experiments and easier viewing of all experiment values from within the solvers. The project-level improvements allow models to be copied within projects and between projects, and a new project interface supports navigation between editors in a model. Editors for graphical formalisms (e.g., SANs and Rep/Join) support snap grids, find/replace, and component copying. In general, the Möbius implementation continues to mature, both through correction of many reported deficiencies in the implementation, and through support of newer versions of software used by Möbius, such as GNU's C++ compiler or Java.

V. Acknowledgments

We would like to acknowledge the work done by several researchers who have collaborated with us on integrating new formalisms and techniques into Möbius. 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.

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, January 2000.

[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 Transactions on Software Engineering*, vol. 28, no. 10, pp. 956–969, October 2002.

[4] W. H. Sanders and J. F. Meyer, "Stochastic activity networks: Formal definitions and concepts," in *Lectures on Formal Methods and Performance Analysis, First EEF/Euro Summer School on Trends in Computer Science*, ser. Lecture Notes in Computer Science, E. Brinksma, H. Hermanns, and J. P. Katoen, Eds., vol. 2090. Berg en Dal, The Netherlands: Springer, 2001, pp. 315–343.

[5] ——, "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, pp. 25–36, Jan. 1991.

[6] W. D. Obal II, "Measure-adaptive state-space construction methods," Ph.D. dissertation, University of Arizona, 1998.

[7] 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*, A. Avizienis, H. Kopetz, and J. Laprie, Eds. Springer-Verlag, 1991, pp. 215–237.

[8] W. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[9] J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[10] P. D'Argenio, H. Hermanns, J.-P. Katoen, and J. Klaren, "MoDeST: A modelling and description language for stochastic timed systems," in *Process Algebra and Probabilistic Methods*, ser. Lecture Notes in Computer Science, L. de Alfaro and S. Gilmore, Eds., vol. 2165. Aachen, Germany: Springer, 2001, pp. 87–104.

[11] H. Bohnenkamp, T. Courtney, D. Daly, S. Derisavi, H. Hermanns, J.-P. Katoen, J. Klaren, V. V. Lam, and W. H. Sanders, "On integrating the Möbius and MoDeST modeling tools," in *Proceedings of the 2003 International Conference on Dependable Systems and Networks*, San Francisco, CA, 2003.

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

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

[14] S. Derisavi, P. Kemper, W. H. Sanders, and T. Courtney, "The Möbius state-level abstract functional interface," *To be published in Perf. Eval.*

[15] S. Derisavi, H. Hermanns, and W. H. Sanders, "Optimal state-space lumping in Markov chains," *To be published in Inf. Proc. Let.*

[16] A. Srinivasan, T. Kam, S. Malik, and R. Brayton, "Algorithms for discrete function manipulation," in *Proceedings of the Int'l Conf. on CAD*, 1990, pp. 92–95.

[17] 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.