MULTI-FORMALISM AND MULTI-SOLUTION-METHOD MODELING FRAMEWORKS: THE MÖBIUS APPROACH

William H. Sanders*, Tod Courtney*, Daniel Deavours**,
David Daly*, Salem Derisavi*, and Vinh Lam*

*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.
{whs, tod, ddaly, derisavi, lam}@crhc.uiuc.edu
http://www.crhc.uiuc.edu/PERFORM

**Information & Telecommunications Technologies Center
University of Kansas
2335 Irving Hill Rd.
Lawrence, KS 66045-7612
deavours@ittc.ku.edu
http://www.ittc.ku.edu/~deavours/

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 to interact with models expressed in other formalisms. This chapter provides an overview of selected performance/dependability modeling tools motivating the need for multi-formalism and multi-solution-method tools, provides an overview of the Möbius multi-formalism, multi-solution modeling tool, and describes new enhancements to the tool by us and others. Möbius 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 hope that this chapter makes a clear case for multi-formalism, multi-solution-method modeling tools, and inspires others to join us in the creation of them.

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 diffi-

cult, 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 aspect 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 (i.e., monolithic) model. A framework should also support multiple modeling languages (i.e., formalisms), 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, with reasonably little effort, 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. History

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. Before outlining our approach to providing these capabilities, we provide a brief review of tool developments that led to this possibility. 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^{+} [36], which uses dynamic queuing networks as its model specification method; HIT [6], which uses a homogeneous, structured paradigm for specifying systems; LQNS [29], which uses layered queuing networks as its specification language; QNAP2 [58], which allows users to specify a model as a network of service stations; and RESQ and RESQME [12], which use extended queuing 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 their extensions. There are many tools in this category; for a comprehensive list, see [15], [56]. The tools include APNN Toolbox [10], DSPNexpress [39], GreatSPN [13], QPN-Tool and HiQPN-Tool [2], SPNL [32], SPNP [20], SURF-2 [7], TimeNET [33], and *UltraSAN* [49], 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 [34], Figaro [9], HARP [5], HIMAP [53], and SAVE [35], which all focus on evaluating the dependability of fault-tolerant computing systems. They also include SPE•ED^{TM} [51], which aims to aid in software performance engineering, and TANGRAM-II [11], which evaluates computer and communication systems using analytic/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 were 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 [50]. We are aware of three tools that take this approach. The first, called IMSE (Integrated Modeling Support Environment) [43], 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) [31],[30], 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 [57], 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 [44], [45]. 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 queuing 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 [16], [17], [18]. SMART supports the analysis of models expressed as stochastic Petri nets and queuing 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 or by sharing actions. Like SHARPE, SMART is an important step forward in that it permits the solution of models made up of multiple submodels.

The DEDS (Discrete Event Dynamic System) toolbox [1], [3] also integrates multiple modeling formalisms into a single software environment, but does so very differently from the previous two tools. In particular, the DEDS toolbox converts models expressed in different modeling formalisms (including queuing networks, generalized stochastic Petri nets, and colored Petri nets) into a common "abstract Petri net notation" [1], [4]. 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. This approach has guided us in the development of the Möbius modeling framework, which we describe in the next section.

3. Möbius Overview

The Möbius framework is an environment for supporting multiple modeling formalisms. In order for a formalism to be compatible with the framework, it must be possible to translate any formalism model into an equivalent model that uses Möbius framework components. Since models are constructed in the specific formalisms, the expressive advantages of the particular formalisms are preserved. Because all models are transformed into framework components, all models and solution techniques in the framework are able to interact with each other. The framework is also extensible, allowing new formalisms and solvers to be added with little impact on existing ones, since new formalisms and solvers communicate using framework components.

In order to accomplish the desired goal of extensibility, framework components must be general enough to express a variety of different formalism components. However, there is a subtle but important point concerning the Möbius framework: it is not meant to be a universal formalism. While formalisms may express only a subset of what is possible within the framework, we believe that the subsets expressed by formalisms are carefully chosen by experienced researchers to accomplish various purposes.

3.1. Framework Components

In order to define the framework, we must identify and abstract the common concepts found in most formalisms. We also must generalize the process of building and categorizing models. We divide the model construction process into several steps. Each step in the process generates a new type of model. The illustration shown in Figure 1 highlights the various model types and other components within the Möbius framework.

The first step in the model construction process is to generate a model using some formalism. This most basic model in the framework is called an *atomic model*, and is made up of state variables, actions, and properties. State variables (for example, places in the various stochastic extensions to Petri nets, or queues in queuing networks) hold state information about a model, while actions (such as transitions in SPNs or servers in queuing networks) are the mechanism for changing model state.

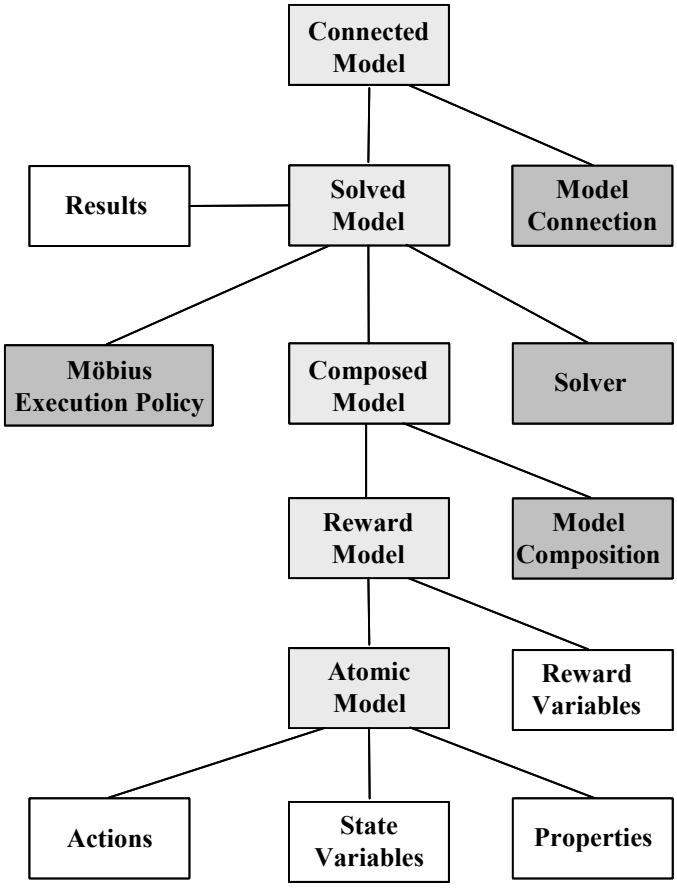

Figure 1 Möbius framework components

Properties provide information about a model that may be needed to allow use of a specialized solver, or to make the solution process more efficient for some solvers.

After an atomic model is created, frequently the next step is to specify some measures of interest on the model using some reward specification formalism, e.g., [47]. The Möbius framework captures this pattern by having a separate model type, called *reward models*, that augments atomic models

with reward variables. Some formalisms may have the measure specification as part of the formalism description. In that case, the formalism produces a reward model instead of an atomic model.

If the model being constructed is intended to be part of a larger model, then the next step is to *compose* it with other models to form a larger model. This is sometimes used as a convenient technique to make the model modular and easier to construct; at other times, the ways that models are composed can lead to efficiencies in the solution process. Examples include the Replicate/Join composition formalism [46], [25] and the graph composition formalism of [40], in which symmetries may be detected and state lumping may be performed as a result of models sharing a subset of their state variables. Another notable composed model technique is synchronization on actions, which is found, for example, in stochastic process algebras (SPAs) such as PEPA [37], as well as in stochastic automata networks (and also SANs, e.g., [42]) and superposed GSPNs (e.g., [27], [38]). Although a composed model is a single model with its own state space, it is not a "flat" model. It is hierarchically built from submodels, and those submodels are minimally impacted by the composition. Note that the compositional techniques do not depend on the particular formalism of the atomic models that are being composed, provided that any necessary requirements are met. Composition of different formalisms (specifically SAN and PEPA) is demonstrated by example models included in the Möbius software distribution.

The next step is typically to apply some solver to compute a solution and generate a *solved model*. We call any mechanism that calculates the solution to reward variables a *solver*. The calculation method could be exact, approximate, or statistical, and it may take advantage of model properties that are due to the atomic model formalisms, model composition, and reward specification. Note that solvers operate on framework components, not formalism components. Consequently, a solver may operate on a model independent of the formalism in which the model was constructed, so long as the model has the properties necessary for the solver.

The computed solution to a reward variable is called a *result*. Since the reward variable is a random variable, the result is expressed as some characteristic of a random variable. This may be, for example, the mean, variance, or distribution of the reward variable. The result may also include any solver-specific information that relates to the solution, such as any errors, the stopping criterion used, or the confidence interval. A solution calculated in this way may be the final desired measure, or it may be an intermediate step in further computation. If a result is intended for further computation, then the result may capture the interaction between multiple reward models that together form a connected model.

A *connected model* is an ordered set of reward models and their corresponding solution methods in which input parameters to some of the models depend on the results of other models in the set. This is useful for modeling using decompositional approaches, such as that used in [19]. In those cases, the model of interest is a set of reward models with dependencies expressed through results, where the overall model may be solved through a system of nonlinear equations (if a solution exists).

3.2. Tool Description

The Möbius tool is our implementation of the Möbius framework. It ensures that formalism components are translated into framework components through the use of the model abstract functional interface (AFI) [28]. The model AFI provides the common interface between model formalisms and solvers that allows formalism-to-formalism and formalism-to-solver interactions. It uses abstract classes to implement Möbius framework components. The model AFI is built from three main base classes: one for state variables, one for actions, and one that defines overall atomic model

behavior. Each of these classes defines an interface used by the Möbius tool when building composed models, specifying reward variables, and solving models. There are subtle differences between the current AFI and the framework, predominantly due to delays in the implementation of newer framework concepts (such as properties), and additions to the AFI to increase efficiency or ease of use of the tool.

The various components of a model formalism must be presented as classes derived from the Möbius model AFI classes in order to be implemented in the Möbius tool. Other model formalisms and model solvers in the tool are then able to interact with the new formalism by accessing its components through the Möbius abstract class interfaces.

The main user interface for the Möbius tool presents a series of editors that are classified according to model type. Each formalism or solver supported by Möbius has a corresponding editor in the main interface. These editors are used to construct and specify the model, possibly performing some formalism-specific analysis and property discovery, and to define the parameters for the solution techniques. The tool dynamically loads each formalism-specific editor from a java archive (jar file) at startup. This design allows new formalisms and their editors to be incorporated into the tool without modification or recompilation of the existing code, thus supporting the extensibility of the Möbius tool.

Models can be solved either analytically/numerically or by simulation. From each model, C++ source code is generated and compiled, and the object files are packaged to form a library archive. These libraries are linked together along with the tool's base libraries to form the executable for the solver. The executable is run to generate the results. The base libraries implement the components of the particular model formalism, the AFI, and the solver algorithms. The organization of Möbius components to support this model construction procedure is shown in Figure 2.

We believe that the majority of modeling techniques can be supported within the Möbius framework and tool. By making different modeling processes (such as adding reward variables, composing, solving, and connecting) modular, we can maximize the amount of interaction allowable between these processes. That approach also makes it possible for the framework to be extensible, in that new atomic modeling formalisms, reward formalisms, composition formalisms, solvers, and connection formalisms may be added independently. Several of these features will be discussed in more detail in the next section, in which we focus on recent enhancements to Möbius.

The Möbius tool originally supported the following formalisms and capabilities:

SANs: The stochastic activity network atomic formalism [47].

Replicate/Join Formalism: A composition formalism that composes models using two constructs: Replicate and Join [46].

Graph Composer: A composition formalism that allows the construction of a composed model as an arbitrary graph of submodels with shared state variables [40].

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 [47].

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.

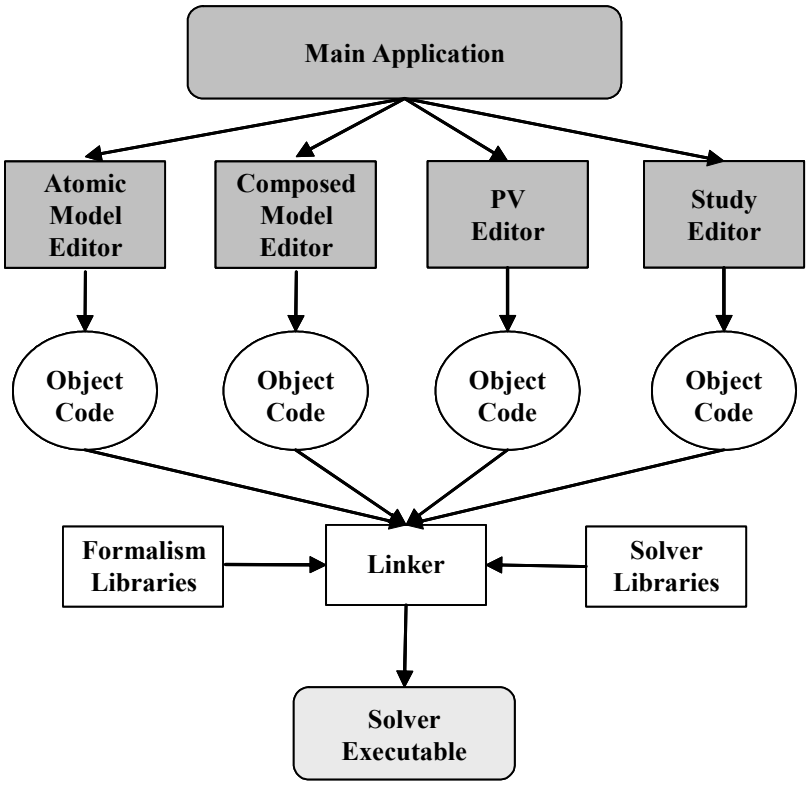

Figure 2 Möbius tool architecture

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.

More detail on the original version of Möbius can be found in [23].

4. Recent Möbius Enhancements

In the process of developing the first Java interfaces for Möbius, 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 Möbius 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.

4.1. 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 basic types of state data (i.e., int, short, char, float, and double) 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.

4.2. Atomic Model Formalisms

The model 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 since the initial model AFI and initial set of formalisms were developed [21], 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, without need for changes to those parts.

Buckets and Balls (B&B [54]) 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 can. Among these extensions are arbitrary cardinality and distribution functions 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) [37] 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 [21], [8] 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. Syntactically, 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.

APNN Toolbox [10] is a tool that aims to create an open toolset around a common exchange interface, the so-called Abstract Petri Net Notation (APNN). The toolbox provides state-space-based analysis methods in which the common state-space explosion problem is mitigated by use of

Kronecker representation techniques. Integration of the APNN Toolbox editor into the Möbius tool at the model-AFI level allows modelers to solve models expressed in the toolbox using Möbius solvers, and to compose APNN models with other Möbius models. A planned integration using the state-level AFI will allow Möbius models to use APNN Toolbox numerical solvers.

The four atomic formalisms summarized in this section represent integrated formalisms within the Möbius tool. While it is often appropriate to fully integrate formalisms into Möbius, there are cases in which 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 that information, Möbius can use the externally defined atomic model within a composed model or reward definition as if it had been defined within Möbius.

4.3. 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 [14]. 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 engines. The lowest level is the virtual database layer, which contains all interaction with the external database. Möbius currently supports the PostgreSQL database engine, and it should be possible to add support for different database engines 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) [59]. 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. 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* [14]. 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.

4.4. 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 [25], 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 re-implement 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 used with one another.

2) Optimal Lumping: In [23], we improved upon existing algorithms for constructing the optimal lumping quotient of a finite Markov chain and presented one with a time complexity of $O(m \lg n)$, 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: We have also extended previous work on Multi-valued Decision Diagrams (MDDs) [54] and Matrix Diagrams (MDs) to composed models that share state variables [25]. 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 [46]) in the form of an MDD and an MD data structure, respectively.

4.5. 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 manager, as well as other assorted improvements. The networking and I/O routines of the simulator have been redesigned to support dis-

tributed 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. 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.

5. CONCLUSIONS

This chapter argues that multi-formalism and multi-solution-methods modeling tools are needed, and describes a tool of this type called Möbius. Möbius supports multiple formalisms and solution methods by having a general and extensible state variable typing system that accommodates a large number of formalism state variables and by having an execution policy and actions generalize the execution policies of most formalisms. Möbius also has properties that preserve information that is important for efficient solution. The Möbius tool implements the framework's capabilities using model- and state-level AFIs. Using these AFIs, models expressed in possibly heterogeneous formalisms can interact with other models and solvers, yielding a tightly integrated environment. New formalisms and solvers are currently under development by us and others. We appreciate the participation of others in this endeavor so far, and welcome participation by others in the future.

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 symbolic state-space generation, Kai Lampka for his work on action synchronization-based composition, and Carsten Tepper for his integration of the APNN Toolbox with Möbius. We would also like to acknowledge the work done by former members of the Möbius group: Amy L. Christensen, Graham Clark, Jay M. Doyle, G. P. Kavanaugh, John M. Sowder, Aaron J. Stillman, Patrick G. Webster, and Alex L. Williamson. We would also like to thank the two newest members of the Möbius group, Shravan Gaonkar and Mark Griffith, for their initial contributions to the tool. Finally, we would like to thank Jenny Applequist for her support of the PERFORM modeling group, in general, and the editing of this chapter, in particular.

References

[1] BAUSE, F., and BEILNER, H. (eds.), Performance Tools: Model Interchange Formats, Forschungsbericht Nr. 581 des Fachbereichs Informatik der Universität Dortmund, Germany, 1995.

[2] BAUSE, F., BUCHHOLZ, P., and KEMPER, P., 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

[3] BAUSE, F., BUCHHOLZ, P., and KEMPER, P., 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.

[4] BAUSE, F., KEMPER, P., and KRITZINGER, P., Abstract Petri Net Notation, in: Petri Net Newsletter, no. 49, pp. 9-27, 1995.

[5] BAVUSO, S. J., BECHTA DUGAN, J., TRIVEDI, K. S., ROTHMANN, E. M., and SMITH, W. E., Analysis of Typical Fault-Tolerant Architectures Using HARP, in: IEEE Trans. on Reliability, vol. 36, no. 2, pp. 176-185, 1987.

[6] BEILNER, H., MÄTER, J., and WEIßENBERG, N., 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.

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

[8] BOHNENKAMP, H., COURTNEY, T., DALY, D., DERISAVI, S., HERMANNS, H., KATOEN, J.-P., KLAREN, R., LAM, V. V., and SANDERS, W. H., On Integrating the Möbius and MoDeST Modeling Tools, in: Proceedings of the 2003 International Conference on Dependable Systems and Networks, San Francisco, CA, 2003, p. 671.

[9] BOUISSOU, M., 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.

[10] BUCHHOLZ, P., FISCHER, M., KEMPER, P., and TEPPER, C., New Features in the APNN Toolbox, in: Tools of Aachen 2001, Intl. Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems (P. Kemper, ed.), Tech report No. 760/2001, pp. 62-67. Universität Dortmund, FB Informatik, 2001.

[11] CARMO, R. M. L. R., DE CARVALHO, L. R., DE SOUZA E SILVA, E., DINIZ, M. C., and MUNTZ, R. R. R., 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.

[12] CHANG, K. C., GORDON, R. F., LOEWNER, P. G., and MACNAIR, E. A., The RESearch Queueing Package Modeling Environment (RESQME), IBM Research Report RC-18687, Yorktown Heights, New York, Feb. 1993.

[13] CHIOLA, G., FRANCESCHINIS, G., GAETA, R., and RIBAUDO, M., 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.

[14] CHRISTENSEN, A. L., Result Specification and Model Connection in the Möbius Modeling Framework, Master's thesis, University of Illinois at Urbana-Champaign, 2000.

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

[16] CIARDO, G., JONES, R. L., MINER, A. S., and SIMINICEANU, R., Logical and Stochastic Modeling with SMART, in: Proceedings of the 13th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation (Performance Tools '03) (ed. by P. Kemper and W. H. Sanders), LNCS 2794, Urbana, IL, USA, September 2003, pp. 78-97.

[17] CIARDO, G. and MINER, A. S., 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.

[18] CIARDO, G. and MINER, A. S., 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.

[19] CIARDO, G. and TRIVEDI, K. S., A Decomposition Approach for Stochastic Reward Net Models, in: Performance Evaluation, vol. 18, pp. 37{59, 1993.

[20] CIARDO, G. and TRIVEDI, K. S., 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.

[21] CLARK, G., COURTNEY, T., DALY, D., DEAVOURS, D., DERISAVI, S., DOYLE, J. M., SANDERS, W. H., and WEBSTER, P. G., The Möbius Modeling Tool, in: Proceedings of the 9th International Workshop on Petri Nets and Performance Models, Aachen, Germany, September 11-14, 2001, pp. 241-250.

[22] D'ARGENIO, P., HERMANNS, H., KATOEN, J.-P., and Klaren, R., 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.

[23] DEAVOURS, D. D., CLARK, G., COURTNEY, T., DALY, D., DERISAVI, S., DOYLE, J. M., SANDERS, W. H., and WEBSTER, P. G., The Möbius Framework and Its Implementation, in: IEEE Transactions on Software Engineering vol. 28, no. 10, October 2002, pp. 956-969.

[24] DERISAVI, S., HERMANNS, H., and SANDERS, W. H., Optimal State-space Lumping in Markov Chains, in: Information Processing Letters, vol. 87, no. 6, pp. 309-315.

[25] DERISAVI, S., KEMPER, P., and SANDERS, W. H., Symbolic State-space Exploration and Numerical Analysis of State-sharing Composed Models, in: Proceedings of NSMC '03: The Fourth International Conference on the Numerical Solution of Markov Chains, Urbana, IL, USA, September 3-5, 2003, pp. 167-189.

[26] DERISAVI, S., KEMPER, P., SANDERS, W. H., and COURTNEY, T., The Möbius State-level Abstract Functional Interface, in: Performance Evaluation special issue on TOOLS 2002, to appear.

[27] DONATELLI, S., Superposed Generalized Stochastic Petri Nets: Definition and Efficient Solution, in: Application and Theory of Petri Nets 1994, LNCS 815 (Proc. 15th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain), R. Valette, Ed., pp. 258{277. Springer-Verlag, June 1994.

[28] DOYLE, J. M., Abstract Model Specification Using the Möbius Modeling Tool, M.S. thesis, University of Illinois at Urbana-Champaign, Jan. 2000.

[29] FRANKS, R. G., HUBBARD, A., MAJUMDAR, S., NEILSON, J. E., PETRIU, D. C., ROLIA, J., and WOODSIDE, C. M., A Toolset for Performance Engineering and Software Design of Client-Server Systems, in: Performance Evaluation, vol. 24, no. 1-2, pp. 117-135, Nov. 1995.

[30] FRICKS, R., HIREL, C., WELLS, S., and TRIVEDI, K., 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.

[31] FRICKS, R., HUNTER, S., GARG, S., and TRIVEDI, K., 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.

[32] GERMAN, R., 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.
[33] GERMAN, R., KELLING, C., ZIMMERMANN, A., and HOMMEL, G., TimeNET: A Toolkit for Evaluating Non-Markovian Stochastic Petri-Nets, in: Performance Evaluation, vol. 24, pp. 69-87, 1995.
[34] GOSWAMI, K. K. and IYER, R. K., DEPEND: A Simulation-Based Environment for System Level Dependability Analysis, in: IEEE Trans. on Computers, vol. 46, no. 1, pp. 60-74, Jan. 1997.
[35] GOYAL, A., CARTER, W. C., DE SOUZA E SILVA, E., LAVENBERG, S. S., and TRIVEDI, K. S., The System Availability Estimator, in: Proc. of the 16th Int. Symp. on Fault-Tolerant Computing (FTCS-16), 1986, pp. 84-89.
[36] HAVERKORT, B. R., Performability Evaluation of Fault-Tolerant Computer Systems Using DyQN-tool+, in: Int. Journal of Reliability, Quality, and Safety Engineering, vol. 2, no. 4, pp. 383-404, 1995.
[37] HILLSTON, J., A Compositional Approach to Performance Modelling, Cambridge University Press, Cambridge, 1996.
[38] KEMPER, P., Numerical Analysis of Superposed GSPNs, in: Proc. Sixth International Workshop on Petri Nets and Performance Models (PNPM '95), Durham, North Carolina, Oct. 1995, pp. 52-61.
[39] LINDEMANN, C., REUYS, A., and THÜMMLER, A., 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.
[40] OBAL, W. D., II, Measure-adaptive State-space Construction Methods, Ph.D. dissertation, University of Arizona, 1998.
[41] OBAL,W. D., II and SANDERS, W. H., Measure-adaptive State-space Construction Methods, in: Performance Evaluation, vol. 44, pp. 237{258, Apr. 2001.
[42] PLATEAU, B. and ATIF, K., A Methodology for Solving Markov Models of Parallel Systems, in: IEEE Journal on Software Engineering, vol. 17, no. 10, pp. 1093-1108, 1991.
[43] POOLEY, R. J., 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.
[44] SAHNER, R. A., Combinatorial-Markov Method of Solving Performance and Reliability Models, Ph.D. Dissertation, Duke University, 1986.
[45] SAHNER, R. A., TRIVEDI, K. S., and PULIAFITO, A., Performance and Reliability Analysis of Computer Systems, An Example-Based Approach Using the SHARPE Software Package, Boston, MA: Kluwer, 1996.
[46] SANDERS, W. H. and MEYER, J. F., Reduced Base Model Construction Methods for Stochastic Activity Networks, in: 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.
[47] SANDERS, W. H., and MEYER, J. F., 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.

[48] SANDERS, W. H. and MEYER, J. F., A Unified Approach for Specifying Measures of Performance, Dependability, and Performability, in: Dependable Computing for Critical Applications, A. Avizienis, J. Kopetz, and J. Laprie, Eds., vol. 4 of Dependable Computing and Fault-Tolerant Systems, pp. 215-237. Heidelberg: Springer-Verlag, 1991.

[49] SANDERS, W. H., OBAL, W. D., II, QURESHI, M. A., and WIDJANARKO, F. K., The *UltraSAN* Modeling Environment, in: Performance Evaluation, vol. 24, no. 1, Oct.-Nov. 1995, pp. 89-115.

[50] SMITH, C. U., 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.

[51] SMITH, C. U. and WILLIAMS, L. G., Performance Engineering Evaluation of Object-Oriented Systems with SPE•ED^{TM}, 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. 135-154.

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

[53] SRIDHARAN, M., RAMASUBRAMANIAN, S., and SOMANI, A. K., 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.

[54] SRINIVASAN, A., KAM, T., MALIK, S., and BRAYTON, R., Algorithms for Discrete Function Manipulation, in: Proceedings of the Int'l Conf. on CAD, 1990, pp. 92–95.

[55] STEWART, W., Introduction to the Numerical Solution of Markov Chains. Princeton University Press, 1994.

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

[57] VAN MOORSEL, A. P. A. and HUANG, Y., 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.

[58] VERAN, M., and POTIER, D., 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.

[59] WEBSTER, P. G., Design of Experiments in the Möbius Modeling Framework, Master's thesis, U. of Illinois at Urbana-Champaign, 2002.