OVERVIEW: AN INTEGRATED FRAMEWORK FOR PERFORMANCE ENGINEERING AND RESOURCE-AWARE COMPILATION[1]

William H. Sanders, Constantine Polychronopoulos, Thomas Huang,
Tod Courtney, David Daly, Dan Deavours, and Salem Derisavi

Coordinated Science Laboratory, Beckman Institute for Advanced Science and Technology,
and Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA
whs@crhc.uiuc.edu, cdp@csrd.uiuc.edu, huang@ifp.uiuc.edu
www.crhc.uiuc.edu/PERFORM/NSF.html

Abstract— Design of next-generation computing and communication systems will be application-driven, and requires fundamental advances in 1) performance engineering frameworks, methods, and tools, and 2) adaptive compilation and runtime support techniques. Our work is taking a systematic and synergetic approach to developing both of these capabilities, and is demonstrating their use via application to several important distributed applications. In conducting the work, we are making fundamental advances in techniques for system and model composition, multi-level/formalism modeling and performance evaluation, adaptive compilation, and dynamic runtime support.

1. Introduction and Motivation

Next-generation parallel and distributed computing systems have the potential to provide greatly increased capabilities and performance. This potential has enabled the development of increasingly complex scientific and commercial applications, which have stringent performance requirements, must be extensible, and must scale well as problem size and/or desired degree of parallelism increases. The significant development costs, maintenance costs, and performance requirements of these applications call for a comprehensive and integrated approach to application composition and development, system and application modeling and evaluation, performance characterization, compiler optimization, and low-overhead runtime support. In order to achieve these capabilities, our research has been pursuing fundamental advances in 1) methods for hierarchical, multi-language modeling, simulation, and evaluation, and 2) techniques for adaptive, resource-aware compilation and runtime support. In this project, we have been taking a systematic and synergetic approach to making these advances and incorporating them in a performance engineering framework and resource-aware compilation and runtime system. In addition, we have been working to demonstrate the use of the framework/system via application to several important parallel and distributed multimedia, video database, and computer vision applications.

[1] This work was supported, in part, by NSF contract number EIA 99-75019.

2. Project Overview

In this paper, we first provide an overview of the project, highlighting work done in each of the three main project areas (Section 2). We then, in Section 3, take a more detailed look at the portion of the project related to performance engineering, describing the Möbius modeling tool that has been built. We conclude in Section 4, briefly describing our plans for future work on Möbius.

2.1 Performance Engineering Framework

Despite the development of many modeling formalisms and model solution methods, most tool implementations developed in the past support only a single formalism. Furthermore, models expressed in the chosen formalism cannot be combined with models expressed in other formalisms. This monolithic approach both limits the usefulness of such tools to practitioners, and hampers modeling research, since it is difficult to compare new and existing formalisms and solvers.

In response to these limitations, we have developed the Möbius modeling framework, which is a performance engineering framework for complete distributed and parallel computing systems that accounts for system components including the application software itself, the operating system, and the underlying computing and communication hardware. The framework is usable in a stand-alone fashion, or in conjunction with the compilation and runtime support environment (after an application and hardware have been developed) to guide the compiler and runtime support system in making decisions concerning specific optimizations and resource allocations. It provides a means by which multiple, heterogeneous models can be composed together, each representing a different module (software or hardware), component, or view of the system. The developed composition techniques permit the models to interact with one another by sharing state, events, or results, and is scalable, in the sense that the solution of the entire model will be possible at a lower cost than for an equivalent unstructured model. Multiple modeling languages have been developed, as well as methods to combine models at different levels of resolution. Finally, we have developed 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.

2.2 Resource-Aware Compiler

In to response the second project research objective, we have developed a unified and extensible compilation system that can be easily reconfigured to automatically parallelize and optimize programs written in imperative languages and for a variety of system architectures, from embedded processors to clusters of workstations. Another objective of this work is to extend and enhance traditional approaches to compiler design in order to support compiler extensibility. An extensible compiler would also provide the means for rapid prototyping of compiler optimizations for various target architectures, and facilitate the development of new analysis or optimization algorithms with minimal effort, using existing source base.

The released version of PROMIS includes many standard global optimizations (including common subexpression elimination, constant folding, copy propagation, dead code removal, and loop invariant code motion) as well as many key loop transformations (including loop fusion, loop distribution, loop interchange, loop peeling, and loop unrolling). Additionally, PROMIS provides induction variable analysis and subscript analysis to improve global data dependence information. One of the principal research contributions of this work is a unique framework for maintaining symbolic analysis information throughout the compilation phases. By supporting callback functions and a standard interface to symbolic environments, this framework is able to enable dynamic and incremental analysis and optimization pass ordering. To the best of our knowledge, PROMIS is the only compiler framework designed around a symbolic analysis core.

High-level information available in the PROMIS UMD enables adaptive parallelization of programs at different levels of thread granularity. This unique feature is ideal for retargeting parallel code to hierarchical parallel architectures. At any given level of the hardware hierarchy (processor-level, node-level, or cluster-level), the UMD provides accurate machine-dependent information about the costs for orchestrating parallelism (including thread creation, deletion, and synchronization, latencies of accesses to different levels of the memory hierarchy, and latencies of communication within and across nodes). The compiler uses the UMD during parallelization passes to adjust the granularity of threads, so that parallelization is profitable and the cost of managing threads does not outweigh the speedup obtained from multithreaded execution.

The project has also undertaken an effort to integrate the UMD, static performance analysis, and runtime support in PROMIS to achieve efficient parallel execution on current and future-generation hierarchical systems.

2.3 Computer Vision: Face Tracking, Video Filtering, and Visualization

Computer vision and video processing (and more specifically, face tracking and analysis and video filtering and indexing) are serving as the primary practical application areas on which we are demonstrating our environment, since they represent the computational cores of diverse scientific and commercial applications (such as automatic object recognition, multimedia and video databases, and data mining), they involve computationally demanding floating point and integer kernels, and they represent application areas of growing importance to the scientific and commercial world. We are testing the framework on two important applications in the computer vision and video processing area: 1) real-time facial detection, motion analysis, and tracking, and 2) video filtering/indexing. Because of their computationally intense nature, widespread importance, and prototypical nature, real-time facial detection, motion analysis and tracking, and video filtering/indexing are applications that can both validate the usefulness of our proposed framework, and benefit from its use. If we are successful, it will be possible to design and implement algorithms for these applications in a much more automated fashion, with predictable performance, and in a manner that permits them to adapt much more easily to different system configurations and resource availabilities. More generally, the compiler/runtime/performance engineering framework will permit the design, with predictable performance, of a variety of complex distributed and parallel applications that are scalable, can run on a variety of different system architectures, and can dynamically adapt to changing resources during their execution. Such an ability would be an important step forward in the development of next-generation software, resulting in better software at a lower cost, and with a shorter development time.

3. Performance Engineering Framework

3.1 Motivation

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 difficult, 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 view 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 model. A framework should also support multiple modeling languages, 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, be 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, without changing existing tool components, to add new modeling formalisms, composition and connection methods, and model solution techniques to a software environment that implements the framework.

3.2 The Möbius Modeling Framework

The Möbius framework is the formal specification of an environment in which multiple modeling formalisms and solvers can interact. As we stated earlier, the goal of the Möbius framework is to maximize the amount of interaction possible between formalisms and solvers.

3.2.1 Framework description

Models are expressed in particular formalisms within the Möbius framework. A modeling formalism, in order to be compatible with the framework, must represent the various model components as framework components. In that way, models of different formalisms and solvers can interact because they are interacting with framework components.

Figure 1 illustrates the relationship between models, formalisms, and solvers within the framework. Users of the Möbius tool enter models in a particular modeling formalism. The formalism editor, in turn, translates each component of the model into a corresponding component of the framework. The user is able to utilize all the advantages of a particular formalism, while at the same time having access to all the other components of the tool, such as various composition, measure, and connection methods, as well as a variety of solvers. This also allows the tool to be extensible, because we can add new formalisms and solvers with a minimum of impact on existing formalisms and solvers.

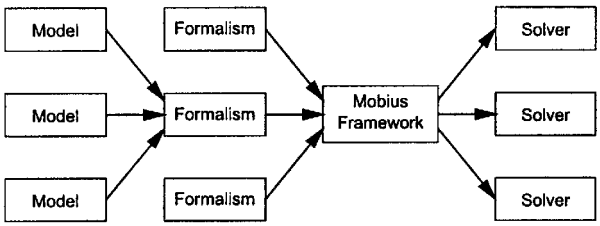

Figure 1. Models, formalisms, and framework.

3.2.2 Framework components

In order to define the framework, we must abstract away many of the concepts found in most formalisms. We also must generalize the process of building and categorizing models. We begin with the illustration shown in Figure 2, which outlines the various components within the Möbius framework.

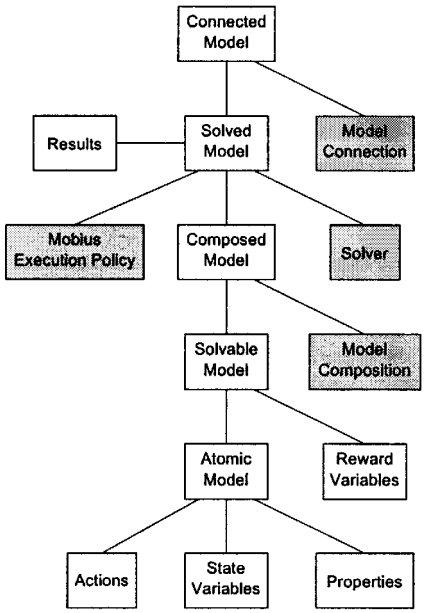

Figure 2. Möbius framework components.

The first step in the model construction process is to generate a model or submodel using some formalism. This most basic model 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 (PN), 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. Properties provide information about a model that may be required in order for a specialized solver to be used, or may be needed

to make the solution process more efficient for some solvers.

After a model or submodel is created, frequently the next step is to specify some measures of interest on the model using some reward specification formalism, e.g., [Sanders 91a]. The Möbius framework captures this pattern by having a separate model type, called *solvable 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 solvable 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 of this include the Replicate/Join composition formalism [Sanders 91] and the graph composition formalism of [Obal 01], in which symmetries may be detected and state lumping may be performed on the fly. Both of those formalisms have been implemented in the Möbius tool [Clark 01]. Other composed model techniques include synchronization on actions, which is found, for example, in stochastic process algebras (SPAs) such as PEPA [Hillston 96], and well as in stochastic automata networks (also SANs, e.g., [Plateau 91]) and superposed GSPNs (e.g., [Donatelli 94, Kemper 95]). 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.

Model composition may preserve or destroy properties of the submodels, and it may add new properties to the resulting composed model. In our framework, solvable models may be composed, and the result of model composition is a composed model. Note that since composition occurs on solvable models, the composed model is also solvable. Furthermore, it is possible to add reward variables to a composed model, provided that they comply with any model compositional properties.

The next step is typically to apply some solver to compute a solution. We call any mechanism that calculates the solution to reward variables a *solver*. The calculation can be exact, approximate, or statistical. It may take advantage of model properties that are due to the atomic model or model composition formalisms. 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 solvable models that together form a connected model.

A *connected model* is a set of solvable models 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 [Ciardo 93]. In those cases, the model of interest is a set of solvable models with dependencies through results, where the overall model may be solved through a system of nonlinear equations (if a solution exists).

3.3 Möbius Tool

In the Möbius tool, the framework is present as an *abstract functional interface* (AFI) that uses abstract classes to implement the Möbius features. While the current AFI does not implement all the framework features described in [Deavours 01] (for historical reasons), it is similar in scope and capability. By using the AFI, other model formalisms and model solvers are then able to interact with the new formalism by accessing its features through the Möbius abstract classes. The set of distilled model features captured in these abstract classes constitutes the AFI.

3.3.1 Atomic Model Formalisms

The Möbius tool supports modeling with any formalism for which a useful mapping to the AFI can be provided. Three atomic model formalisms have already been implemented within the Möbius tool, providing concrete evidence for the claim that Möbius supports heterogeneous modeling. This section gives details of the three atomic model formalisms. We demonstrate how the formalism-independent communication layer, the AFI, is implemented for each of these formalisms.

Stochastic Activity Network Formalism The first formalism implementation is *stochastic activity networks* (SANs [Meyer 85]). SANs are an extension to Petri nets [Bause 96] that allow a modeler to build dependability and performance models. SANs are a high-level modeling formalism, and each model leads to the definition of a stochastic process that describes the behavior of a system. Five SAN primitives are used to specify a model in the SAN formalism: places, activities, input gates, output gates, and arcs. SANs have one type of state variable, called a *place*. Each place holds a portion of the model state as a natural number. *Activities* are SAN primitives that represent actions that take some specified (often nondeterministic) amount of time to complete. Activities come in two varieties: *timed* and *instantaneous*. Instantaneous activities take zero time to complete, whereas the time to completion for timed activities is expressed as a random variable. Activities can be augmented with *cases*, which represent possible outcomes of an activity's completion. Each case is assigned a probability, which can be a function of the model state. Upon activity completion,

an activity case is selected based upon the case probabilities, and the associated case-specific state change is performed.

SAN *input gates* are primitives used to specify both action-enabling and state-changing functions. Each input gate consists of two parts: an *input gate predicate* and an *input gate function*. If an input gate is associated with an activity, then the activity is only enabled if all of its associated input gate predicates are true. Input gate predicates are Boolean expressions that can be functions of the model's state. The input gate function specifies a change of model state that is executed when its associated activity completes. *Output gates* are SAN primitives used to specify additional state functions associated with an activity's completion. These state functions are specified in the *output gate function*. Output gates can also be associated with individual activity cases. The SAN atomic model editor is illustrated in Figure 3.

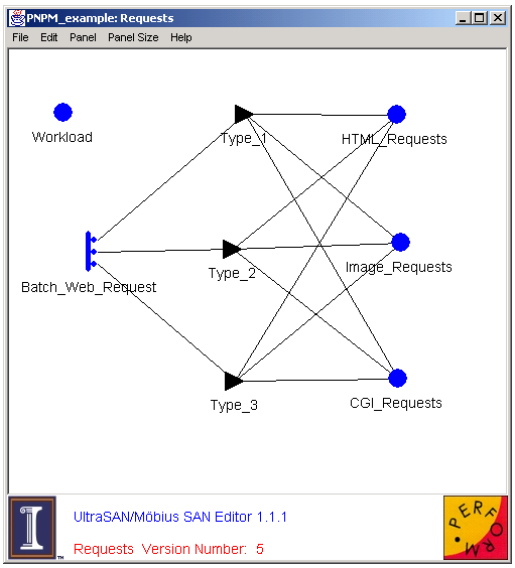

Figure 3. SAN atomic model editor.

Buckets and Balls Formalism The buckets and balls (B&B [Stewart 90]) formalism has also been implemented as an atomic modeling formalism within Möbius. B&B is a fairly simple formalism that defines a number of extensions to Markov chains and makes possible the expression of more complex model behavior than is possible with a Markov chain. Consider a continuous-time Markov chain as a graph whose nodes (buckets) represent the states of the chain, and whose directed arcs are weighted by the rates of the transitions among the states. A single token (ball) is used in the graph to represent the state of the Markov chain; as time passes the ball moves among the buckets, showing changes in the state of the chain.

PEPA Formalism Möbius also supports an alternative style of modeling by providing PEPA (Performance Evaluation Process Algebra, [Hillston 96]), a *stochastic process algebra*, as another atomic model formalism. Mathematically, PEPA models are completely specified as terms of a simple algebra. PEPA extends classical process algebra with the capacity to assign rates to activities, leading to the definition of a stochastic process. PEPA has been applied to modeling the performance of distributed computer systems, and, for example, components of a flexible manufacturing system [Holton 95]. In contrast to the graph-based approach of SANs and B&B, building PEPA models is analogous to writing programs. A picture of the PEPA atomic model editor is shown in Figure 4.

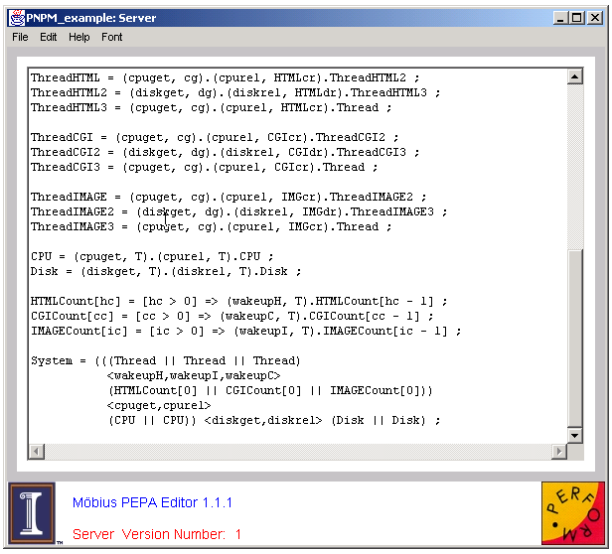

Figure 4. PEPA atomic model editor.

3.3.2 *Composed Models*

Composed models appear at the second layer of the Möbius modeling hierarchy. One way to compose models together is to have two or more models share actions by synchronizing on an action; that is, the action is enabled only if it is enabled in both models, and it completes at the same time in both models.

A second general way to compose models together is by sharing some component of their state. For this reason, the AFI supports state-sharing among models. We can use the concepts of primary and secondary value functions to define how models are constructed through shared state. Sharing state in the AFI requires that a state variable (or a portion of a state variable) in a model be joined to another state variable (or a portion of another state variable) through a sharing relationship.

We now describe the two composition formalisms that have been implemented in Möbius.

Replicate/Join Composition Formalism The Replicate/Join composition formalism, originally conceived for SAN models [Sanders 91], has been implemented as a generic model composition formalism in Möbius. This formalism enables the modeler to define a composed model in the form of a tree, in which each leaf node is a prede-

fined atomic or composed model, and each non-leaf node is classified as either a *Join* or a *Replicate*.

Graph Composition Formalism A second model composition formalism, which we call *graph composition*, is also implemented in the Möbius tool. Graph composition [Obal 98, Stillman 99] is a method to construct composed models that is more general than the Replicate/Join composition method. As with the Replicate/Join composition formalism, the fundamental operation is joining two or more models through sharing a part of the state (which, in the Möbius framework, is abstractly described as one or more state variables).

As described above, the structure of a Replicate/Join composed model is a tree. In contrast, with the graph composition formalism, any model can share its state variables with the state variables of one or more other models.

In order to depict the state-sharing relationship among the models, we use a graph. The vertices of the graph are called *fragments*, and are singleton subsets of the set of all of the state variables of the model. A *private* fragment is distinguished, and it contains all state variables that are not shared by any other model (the private fragment may be empty). The union of all the fragments (including the private fragments) equals the set of the state variables of the model. The private fragment is connected by edges to all the other fragments.

3.3.3 Reward Models

Reward models [Sanders 91a] are present at the third layer of the Möbius tool modeling hierarchy. A reward model defines measures on a submodel, which may be either atomic or composed. The solution of the measures gives the result of solving the model, and tells the modeler about the behavior of the model. At this time we have implemented one type of reward model in Möbius: a *performance variable* (PV^2). A PV allows for the specification of a measure on one or both of the following: 1) the states of the model, leading to a *rate reward* PV; and 2) action completions, leading to an *impulse reward* PV.

A rate reward is a function of the state of the system at an instant of time. An impulse reward is a function of the state of the system and the identity of an action that completes and is evaluated when a particular action completes.

A PV is the combination of a rate reward defined on the model, and an impulse reward (which can be state-dependent or the zero function) defined on each action in the model. The rate and impulse rewards provide instantaneous values, but do not indicate when these values should be measured or accumulated. A performance variable can be specified to be measured at an instant of time, in steady state, accumulated over a period of time, or accumulated over a time-averaged period of time. Once the rate and impulse rewards are defined, the desired statistics on the measure must be specified. The options include solving for the mean, variance, or distribution of the measure, or the probability of the measure falling within a specified range.

[2] Note that although these variables are called "performance variables," they are generic and can be used to represent dependability and performability variables as well.

3.3.4 Studies

During the specification of atomic, composed, and reward models, global variables can be used to parameterize model characteristics. A global variable is a variable that is used in one or more models, but not given a value. In the Möbius tool, global variables can have any of the basic C++ types, including short, int, and double. Models are solved after each global variable is assigned a specific value. One such assignment forms an *experiment*. Experiments can be grouped together to form a *study*. Studies allow model parameters to be varied in an organized manner providing an effective measurement of the reward variable behavior for various sets of input parameter values. The Möbius tool allows the modeler to choose from three types of studies: *Set*, *Range*, and *Design of Experiments*.

Set studies allow vectors of arbitrary fixed values to be assigned to each global variable for each experiment. The Möbius tool allows sets of global variable values to be imported from spreadsheets.

Range studies allow each global variable to be defined either as a fixed value or over a range of values. There are four types of ranges: incremental, functional, manual, and random.

Design of Experiments (DOE [Montgomery 01]) studies differ from other types of studies because in addition to assigning global variable values, DOE studies also take the reward variable solutions as input and use this additional information to analyze how the chosen global variables impact the reward variables. Sensitivity analysis can measure the effects of all model parameters and their interactions on each solved reward variable. In addition, the model parameter values that produce optimal reward variable values can be determined.

3.3.5 Solution Techniques

The Möbius tool currently supports two classes of solution techniques: discrete event simulation, and state-based, analytical/numerical techniques. Any model specified using Möbius may be solved using simulation. Models that have delays that are exponentially distributed, or have at most one concurrently enabled deterministic delay, may be solved using a variety of analytic techniques. We begin by describing the Möbius simulators.

Simulation The Möbius simulator supports two types of simulation: transient and steady-state. Transient simulation is done using the independent replication technique to obtain statistical information about the specified reward variables. It is used for solving for reward measures specified at finite time points or over finite time intervals. Steady-state simulation is done using batch means with deletion of an initial transient to solve for steady-state, in-

stant-of-time variables. The modeler is required to specify the initial transient and the batch size.

Estimates available during simulation include mean, variance, interval, and distributions. The results for mean and variance provide confidence intervals. The Möbius simulator may be executed on a single machines, or distributed on a network of machines or operating systems. This parallelism is accomplished by collecting different observations on different machines in the case of transient simulation, or running different trajectories on different machines in the case of batch means.

Analytical/Numerical Solvers Another solution method available in the Möbius tool is provided by the collection of analytical/numerical solvers. By the use of a *state-space generator*, a model can be converted to a Markov chain or a semi-Markov chain. We ensured that the Möbius tool was able to support the stochastic process formats used by *UltraSAN* [Sanders 95] so that we could compare the results produced by the two tools.

The first step in analytic solution with the Möbius tool is the translation of the model into a stochastic process representation, which involves searching the state space of the model. That is done by the state-space generator. Note that symmetries in the model are detected and employed by the various composition formalisms. In particular, all communication with the state-space generator is done through the AFI, allowing the state-space generator to be employed on any Möbius model. This is important in that it allows the state-space generator to be generic, and not need to understand the semantics of a model for which it is generating a state space.

Once the underlying state space is generated, users can choose from a number of analytical solvers to solve for transient and steady-state behaviors. Steady-state solvers include an SOR solver [Malhis 96] and a deterministic/iterative solver [Tvedt 90]. For transient solution, solvers can compute the mean, variance, and distribution of instant-of-time reward variables [Tvedt 90, van Moorsel 97, van Moorsel 94], and the mean of interval-of-time reward variables using several variants of uniformization. We also have implemented a solver that can compute steady-state solutions to models with at most one concurrently enabled deterministic activity.

A solved model sits at the fourth layer in the Möbius hierarchy. The fifth and final layer supports connected models, which are parameterized on the results of solved models. The Möbius tool contains database technology to facilitate the exchange of results between models that make up a connected model, support the use of DOE studies, and provide a method for organizing and storing results obtained by solving models. The design and implementation of the results database is described next.

3.3.6 Results Database

Each of the solvers in the Möbius tool is integrated with the Möbius results database [Christensen 00]. The results database stores the parameter values that define the run and the results generated by the solvers. It provides its own abstract interface in multiple languages that allows both internal and external clients to retrieve parameters and results of the runs. The database is currently used by the Möbius DOE editor to retrieve results for sensitivity analysis. It will also be used in connection formalisms to retrieve previously computed results as inputs to new experiments. The results database has a hierarchical design consisting of three layers. The highest layer is the Möbius Access layer (MAC). This layer provides methods that operate on Möbius-specific data structures, such as the parameters for a simulator run. Client software uses the methods in the MAC layer to interface with the database. The middle layer is the Virtual Database layer. It provides an abstract interface to the lowest layer, the specific third-party database system. A virtual database interface can be constructed for any specific database system, including relational or object-oriented systems. This allows Möbius to be flexible and support both open source and commercial database software. Currently a Virtual Database layer has been written for the *PostgreSQL* database system [PostgreSQL].

4. Möbius Summary and Status

Through careful definition of a modeling framework and abstract functional interface, we were able to construct a performance engineering framework that is extensible, and supports multiple modeling formalisms and model solution methods. To date, we have implemented three atomic modeling formalisms (stochastic activity networks, buckets and balls, and the PEPA stochastic process algebra), two composition formalisms (Replicate/Join composition and Graph composition), one reward variable specification formalism, and several study editors. Multiple simulation and analytical/numerical solvers have also been implemented. Additional modeling formalisms and solution methods are currently under development by others and us. This list gives evidence for our claim that the AFI makes Möbius extensible. It also bodes well for Möbius's use as a vehicle for others to implement new modeling formalisms and solution methods, and we welcome participation by others in this endeavor. Initial evaluation of the tool by one of our industrial partners also suggests that Möbius meets their needs and contains features that make modeling of large-scale systems practical. We look forward to further feedback from our industrial and academic users.

Now that the framework has been developed and the initial set of formalisms and solvers has been implemented, we plan to focus our efforts on the development of new model composition and connection methods, in an attempt to deal with the longstanding problem of system complexity in modeling. We hope to find ways to efficiently predict the performance and dependability of ever larger and more complex parallel and distributed systems.

ACKNOWLEDGMENTS

We would like to thank Jenny Applequist for her editorial assistance in the production of this paper. We would also like to acknowledge the contributions of the former members of the Möbius group: A. L. Christensen, G. Clark, J. Doyle, G. P. Kavanaugh, J. M. Sowder, A. J. Stillman, P. Webster, and A. L. Williamson.

References

[Bause 96] F. Bause and P. S. Kritzinger, *Stochastic Petri Nets*, Vieweg, Wiesbaden, 1996.

[Christensen 00] A. L. Christensen, "Result Specifcation and Model Connection in the Möbius Modeling Framework," M.S. thesis, University of Illinois at Urbana-Champaign, 2000.

[Ciardo 93] G. Ciardo and K. S. Trivedi, "A Decomposition Approach for Stochastic Reward Net Models," *Performance Evaluation*, vol. 18, pp. 37–59, 1993.

[Clark 01] G. Clark, T. Courtney, D. Daly, D. D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, "The Möbius Modeling Tool," in *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, Aachen, Germany, Sept. 2001, pp. 241-250.

[Deavours 01] D. D. Deavours and W. H. Sanders, "Möbius: Framework and Atomic Models," in *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, Aachen, Germany, Sept. 2001, pp. 251-260.

[Donatelli 94] S. Donatelli, "Superposed Generalized Stochastic Petri Nets: Definition and Efficient Solution." In R. Valette, editor, *Application and Theory of Petri Nets 1994, Lecture Notes in Computer Science 815 (Proc. 15th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain)*, pages 258–277. Springer-Verlag, June 1994.

[Hillston 96] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[Holton 95] D. R. W. Holton, "A PEPA Specifcation of an Industrial Production Cell," *The Computer Journal*, vol. 38, no. 7, pp. 542–551, 1995.

[Kemper 95] P. Kemper, "Numerical Analysis of Superposed GSPNs," in *Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)*, Durham, North Carolina, USA, Oct. 1995, pp. 52-61.

[Malhis 96] L. Malhis and W. H. Sanders, "An Efficient Two-Stage Iterative Method for the Steady-State Analysis of Markov Regenerative Stochastic Petri Net Models," *Performance Evaluation*, vol. 27&28, pp. 583–601, 1996.

[Meyer 85] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic Activity Networks: Structure, Behavior, and Application," in *Proc. of the Int. Conf. on Timed Petri Nets*, Torino, Italy, July 1985, pp. 106-115.

[Montgomery 01] D. Montgomery, *Design and Analysis of Experiments*, John Wiley & Sons, Inc., 5th edition, 2001.

[Obal 98] D. Obal, "Measure-Adaptive State-Space Construction Methods," Doctoral Dissertation, University of Arizona, 1998.

[Obal 01] W. D. Obal II and W. H. Sanders, "Measure-adaptive State-space Construction Methods," *Performance Evaluation*, vol. 44, pp. 237–258, Apr. 2001.

[Plateau 91] B. Plateau and K. Atif, "A Methodology for Solving Markov Models of Parallel Systems," *IEEE Journal on Software Engineering*, vol. 17, no. 10, pp. 1093–1108, 1991.

[PostgreSQL] "PostgreSQL," http://www.postgresql.org.

[Sanders 91] W. H. Sanders and J. F. Meyer, "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.

[Sanders 91a] 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* (ed. A. Avizienis, H. Kopetz, and J. Laprie), Springer-Verlag, 1991, pp. 215-237.

[Sanders 95] W. H. Sanders, W. D. Obal II, M. A. Qureshi, and F. K. Widjanarko, "The *UltraSAN* Modeling Environment," in *Performance Evaluation*, vol. 24, no. 1, pp. 89-115, Oct.-Nov. 1995.

[Stewart 90] W. J. Stewart, *MARCA: MARKOV CHAIN ANALYZER A Software Package for Markov Modelling*, Department of Computer Science, North Carolina State University, Raleigh, N.C. 27695-8206, USA, August 1990, See http://www.csc.ncsu.edu/faculty/WStewart/MARCA/marca.html.

[Stillman 99] A. Stillman, "Model Composition in the Möbius Modeling Framework," M.S. Thesis, University of Illinois, 1999.

[Tvedt 90] J. Tvedt, "Solution of Large-sparse Stochastic Process Representations of Stochastic Activity Networks.," M.S. thesis, University of Arizona, 1990.

[van Moorsel 94] A. P. A. van Moorsel and W. H. Sanders, "Adaptive Uniformization," *ORSA Communications in Statistics: Stochastic Models*, vol. 10, no. 3, pp. 619–648, August 1994.

[van Moorsel 97] A. P. A. van Moorsel and W. H. Sanders, "Transient Solution of Markov Models by Combining Adaptive & Standard Uniformization," *IEEE Transactions on Reliability*, vol. 46, no. 3, pp. 430–440, Sept. 1997.