Reliability and Performability Modeling using SHARPE 2000

C. Hirel, R. Sahner, X. Zang, K. Trivedi
Center for Advanced Computing and Communication
Department of Electrical and Computer Engineering
Duke University, Durham, NC 27708-0291, U.S.A.
{chirel,kst}@ee.duke.edu

The SHARPE package is now 13 years old. A well known package in the field of reliability and performability, SHARPE is used in universities as well as in companies. Many important changes have been made during these years to improve the satisfaction of our users. Recently several new algorithms have been added and a Graphical User Interface has been implemented. This paper presents the current status of the tool.

1 Introduction

Assessment of performance, reliability and availability is a key step in the design, analysis and tuning of computer systems. Suppose we have a multiprocessor system and we want to be sure it provides enough processing power. If we add a processor, how much better will the performance get? Could we get a performance improvement just by changing the scheduling of jobs? How would adding a processor affect the reliability of the system? Would an increase in performance outweigh the decrease in reliability? Determining what questions to ask and what measures will address them involves examining the goals and requirements set out by system designers/users. Performance requirements might be focused more on system throughput, on response time, or on meeting deadlines. That is, is it more important that a certain number of jobs or transactions can be processed per unit time, or that individual jobs can expect a certain average response time, or that all jobs are guaranteed a certain maximum response time? Having decided what measures are needed, a system designer has several options when it comes to predicting their values:

- Make an educated guess based on experience with previous, similar systems.
- Build one or more systems (or prototypes) and take measurements.
- Use discrete-event simulation to model the system.
- Construct analytic models of the system.

These options are not exclusive; a system designer may very well use two or more of them, depending on the stage of the design process, the nature and rigidity of the system requirements, and the time and resources available. Each option brings something to the design process. Analytic models are generally more of an abstraction of the system than a discrete-event simulation model, which means that the results might not be as good predictors of system behavior. Modelers must be very careful to choose good abstractions, and take care in parameterizing the models and validating them. But once an analytic model is set up it is easy and fast to carry out trade-off studies, answer "what if" questions, perform sensitivity analyses and compare design alternatives. A system designer has a wide range of kinds of analytical models to choose from. Each model has its strengths and weaknesses in terms of accessibility, ease of construction, efficiency and accuracy of solution algorithms, and availability of software tools. No single kind of model is best, or even necessarily appropriate, for every system and every measure of interest. A modeler who is familiar with many different kinds of models, can easily choose models that best suit a particular system and the kind of measure that is needed at each stage of the design. It is also possible to use different kinds of models hierarchically for different physical or abstract levels of the system and to use different kinds of models to validate each other's results[3].

The models included in the SHARPE, Symbolic Hierarchical Automated Reliability and Performance Evaluator, engine are:

- combinatorial reliability models: reliability block diagrams, fault trees and reliability graphs;

- directed, acyclic task precedence graphs;
- Markov and semi-Markov models, including Markov reward models;
- product-form queueing networks;
- generalized stochastic Petri nets.

We believe that SHARPE is a useful modeler's "toolchest" because it contains support for multiple model types and provides flexible mechanisms for combining results so that models can be used in hierarchical combinations. SHARPE allows its users to construct and analyze performance, reliability, availability and performability models. It gives users direct and complete access to the model types without making any assumptions about an application domain.

2 SHARPE

SHARPE is a *well known* modeling tool that is installed at over 280 Sites. The package has been ported to most architectures and operating systems. SHARPE combines *Flexibility* of Markov models and efficiency of combinatorial models, and is used for software and hardware reliability/performance/performability modeling.

A high level view of methods can be seen in Figure 1.

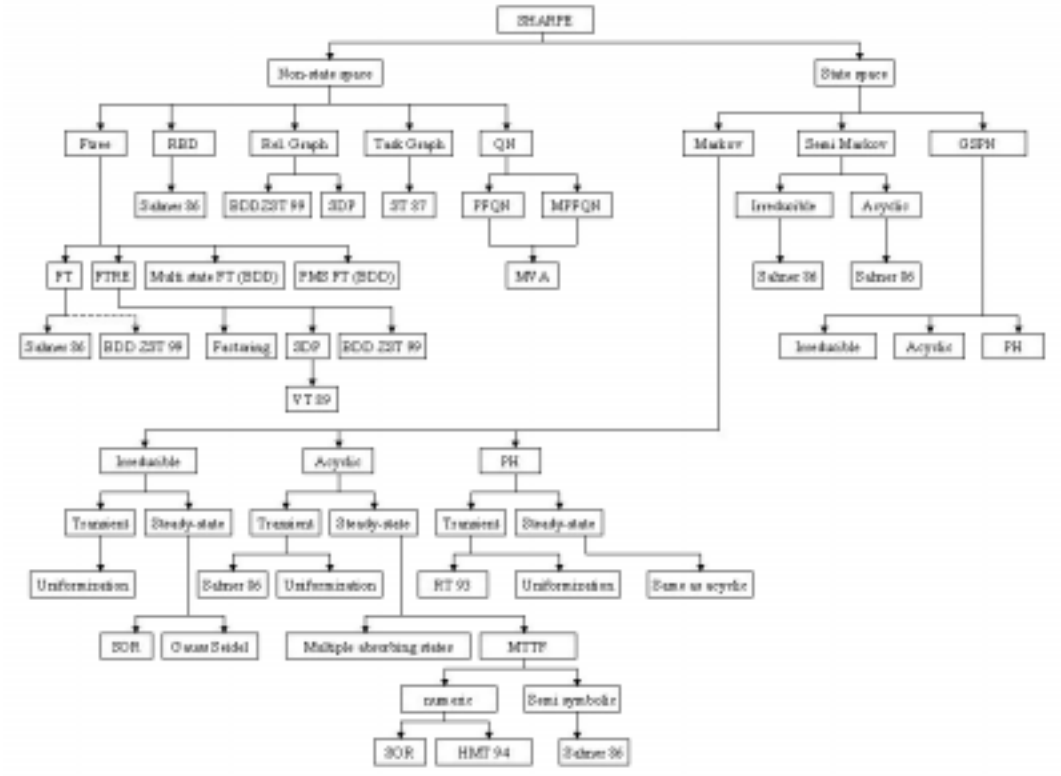

Figure 1: Description of the SHARPE structure

2.1 SHARPE Menu of Model Types

SHARPE is used for dependability, performance and performability. SHARPE models were designed to answer the question: given time-dependent functions that describe the behavior of the components of a system and a description of the structure of the system, what is the behavior of the system as a whole as a function of time? The functions might be cumulative distribution functions (CDFs) for component failure times, CDFs for task completion

times, or the probabilities that components are available at a given time. The system structure might be specified, for example, in the form of a fault tree, a task graph or a Markov chain.

2.1.1 Dependability

The three model types that are commonly used for reliability and availability are reliability block diagrams, fault trees, and reliability graphs. The reliability graph, fault tree and series-parallel block diagram models are similar in that they capture conditions that make a system fail in terms of the structural relationships between the system components.

2.1.2 Performance

The two model types that are commonly used for performance are task graph and product-form queuing network. Graph models are commonly used to study the behavior of programs or processes that contain concurrency and/or probabilistic branching. The models assumed no contention for resources. The queueing network model is useful for examining performance in systems where limited resources must be shared. As the size of a queueing network increases, the state spaces of the associated Markov chain grows much faster. However, there is a useful class of queueing networks that have product form, which means that the joint probability of the queue sizes in the network is a product of the probabilities for the individual service centers. Product-form networks can be analyzed for steady-state measures without having to generate the underlying state space for the whole network.

2.1.3 Both Dependability and Performance

Modeling any system with either a pure performance model or a pure reliability/availability model can lead to incomplete, or, at least, less precise results. Gracefully degrading systems may be able to survive the failure of one or more of their active components and continue to provide service at a reduced level. One of the most commonly used technique for the modeling of gracefully degradable systems is the Markov reward model (MRM). But we may use also the following model types: Markov chains, acyclic or irreducible semi-Markov chains and generalized stochastic Petri nets. SHARPE supports Generalized Stochastic Petri Nets (GSPN) as a specification technique for largeness tolerance; GSPN models are transformed into Markov chains for analysis. The Generalized Stochastic Petri net is the only model type in SHARPE that requires a conversion to a different model (Markov chain) to be solved.

2.1.4 Hierarchical and Hybrid Compositions of Above

Large model tolerance must apply to specification, storage and solution of the model. If the storage and solution problems can be solved, the specification problem can be solved by using more concise (and smaller) model specifications that can be automatically transformed into Markov models (e-g., the GSPN model in SHARPE). Large models can be avoided by using hierarchical model composition. The ability of SHARPE to combine results from different kinds of models makes it possible to use state-space methods for those parts of a system that require them, and use non-state-space methods for the more "well-behaved" parts of the system.

3 Recent additions to SHARPE engine

Several new additions have been made to the SHARPE engine. One of our main concerns is to have the ability to easily construct and analyze larger fault trees and reduce the time needed to solve the models given by the companies using SHARPE. The new requirement of our customers pushed us to optimize some parts in the previous package and to incorporate new features in the new version. The capability to program new algorithms and easily incorporate them into the set of underlying engines is one key in the maintenance of our tool.

1. More built-in distribution functions:

   Over the years, some new built-in functions were added. Early in 1998, a new mechanism was put into SHARPE providing new additional built-in functions like the erlang distribution, the hypoexponential distribution, the hyperexponential distribution, the weibull distribution, the defective distribution: defective, the mixture distribution, the instantaneous component unavailability function etc ...

2. Loop capability in CTMC:

   Early in 1997, SHARPE was modified to support loops in the input language specification of Markov models, including Markov reward models. The loop feature is use to speed up the construction of the Markov chain when a part of this model has a repetitive structure.

3. New functions and algorithms for fault trees and reliability graphs:

   Three algorithms are implemented in SHARPE for fault tree analysis: series-parallel formula (used for fault trees without repeated components [5]), VT algorithm (A multiple inversion (MVI) algorithm to obtain sum of disjoint products (SDP) from mincut set [6]) and the factoring/conditioning algorithm. This last algorithm converts the fault tree with repeated components to a set of fault trees without repeated components by decomposing the repeated components then applying series-parallel formula to theses converted fault tree. Normally this algorithm is very fast when the number of repeated components is less than 14. With the addition of BDD-based algorithm (Binary Decision Diagram), SHARPE can solve very large fault trees. The efficency of BDD algorithm is an improvement of the original VT algorithm. Reliability/unreliability can be calculated from BDD, mincut set can be obtained during the analysis. The event's contribution to the system reliability (importance measures) can also be obtained.

4. New GSPN functions:

   GSPN (generalized stochastic Petri nets) models fall into categories that correspond to the type of the underlying Markov chain: acyclic, irreducible (every state reachable from every other state) and phase-type (containing an irreducible part and absorbing states). When the GSPN model type was first added to sharpe, steady-state measures were provided for all GSPN types and transient measures were provided for acyclic and phase-type GSPNs.

   The transient functions were computed by analyzing the underlying Markov chain for a time-symbolic exponential polynomial form for the desired function, then evaluating the exponential polynomial at time t. Values for additional values of t were computed from the same exponential polynomial. The irreducible GSPN analysis code was extended to make use of this algorithm for the underlying Markov chain. The time-symbolic exponential polynomial forms for irreducible and phase-type GSPNs were made available. For all GSPN types, transient functions were added that were computed for a specified time t numerically using the same "uniformization" algorithm [7] that had been in use for Markov chains for some time. This algorithm is a more stable computation then the one used to produce an exponential polynomial result, but must be done separately for each value of t.

4 Graphical User Interface

The major components of the interface are a model editor, which allows graphical input of the type of models chosen to define the model and an extensive collection of visualization routines to analyze output results of SHARPE. The interface provides a high level input format to the SHARPE syntax which provides great flexibility to users. The interface was defined to minimize the human intervention during the system design process. Careful consideration was given to the design and implementation of the SHARPE interface to facilitate the creation of the models but also the use of the hierarchy feature. The "fault tree" and "reliability block diagrams" designs are defined to speed up the creation of the model. Instead of the use of the "drag and drop" method, the gui generates automatically the objects, for example the gates and events for the fault tree model. The user can define a Markov chain model by using a matrix, thus he can add new nodes and give the rate of any arc. Instead of asking the designer to enter each time the failure rates of the most used components, the interface offers a database, which contains the name of these components and their failure rates. The interface provides a way to plot the results of SHARPE, but it allows also the creation of Excel spreadsheets containing these data.

The interface is designed with Java, which is architecture neutral and portable.

4.1 Status

The model types that are implemented in this version are:

- Fault trees, Reliability block diagrams and Reliability graph
- Markov chains and Generalized stochastic Petri nets

- Product-form queuing networks, Multiple-chain product-form queuing networks and Series-parallel graphs
- Phase-mission system

The paradigms can be intermixed for reliability and performability engineering (hierarchy feature).

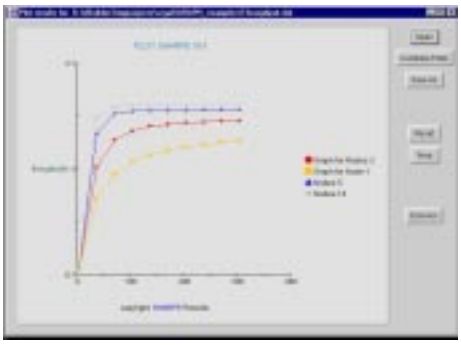

Figure 2: Markov chain model and results given by the interface

References

[1] Kishor S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Sciences Applications.* Prentice-Hall, INC., Englewood Cliffs, New Jersey 07632, 1982.

[2] Gunter Bolch, Stefan Greiner, Hermann de Meer, Kishor S. Trivedi. *Queueing Networks and Markov Chains Modeling and Performance Evaluation with Computer Science Applications.* John Wiley and Sons, New York, NY 10158-0012.

[3] Robin A. Sahner, Kishor S. Trivedi, Antonio Puliafito. *Performance and Reliability Analysis of Computer Systems.* Kluwer Academic Press, 1996

[4] R. A. Sahner. *A Hybrid, Combinatorial-Markov Method of Solving Performance and Reliability Models.* Phd's thesis: Dept. of Comp. Sc., Duke University, Dec. 1985

[5] T. Luo and K.S. Trivedi. *An improved algorithm for coherent-system reliability.* IEEE Transaction on Reliability, 47(1): pp. 73-78, 1998.

[6] J. Muppala. *Numerical Transient Solution of Finite Markovian Queueing Systems.* Queueing and Related Models, U. N. Bhat and I. V. Basawa (ed.), pp. 262-284, Oxford University Press, 1992.

[7] S. Hairong Sun, X. Zang and K.S Trivedi. *A BDD-based Algorithm for Reliability Analysis of Phased-Mission Systems.* IEEE Transactions on Reliability, Vol. 48, No. 1, pp. 50–60, March 1999.

[8] P. Heidelberger, J. Muppala and K.S. Trivedi. *Accelerating Mean Time to Failure Computations.* Performance Evaluation, Vol. 27 & 28, October 1996, North-Holland, pp. 627-645.