SHARPE at the Age of Twenty Two

Kishor S Trivedi
Department of ECE, Duke University
Durham, NC 27708, USA
kst@ee.duke.edu

Robin Sahner
rasahner@gmail.com

ABSTRACT

This paper discusses the modeling tool called SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator), a general hierarchical modeling tool that analyzes stochastic models of reliability, availability, performance, and performability. It allows the user to choose the number of levels of models, the type of model at each level, and which results from each model level are to act as which parameters in which higher-level models. SHARPE includes algorithms for analysis of fault trees, reliability block diagrams, acyclic series-parallel graphs, acyclic and cyclic Markov and semi-Markov models, generalized stochastic Petri nets, and closed single- and multi-chain product-form queueing networks. For many of these, the user can choose among alternative algorithms, and can decide whether to get a result in the form of a distribution function (symbolic in the time variable) or as a mean or probability. SHARPE has been useful to students, practicing engineers, and researchers. In this paper we discuss the history of SHARPE, give some examples of its use, and talk about some lessons learned.

1. INTRODUCTION

Symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE) is a software tool that accepts and solves stochastic models of reliability, availability, performance, and performability. The first version, developed as a part of Robin Sahner's Ph.D. thesis [41] was distributed in 1986. It provided a basic structure for doing analysis of hierarchies of models. The analysis was "semi-symbolic" in the sense that the results were distribution functions symbolic in the time variable t, rather than mean values or probabilities at specific times. The user could decide what model type to use at each level; SHARPE provided a language for specifying which results from lower-level models were to act as which parameters in the higher-level models. SHARPE has catered to three groups of users: practicing engineers, researchers in performance and reliability modeling, and students in science and engineering courses. The first company to adopt SHARPE was Digital Equipment Corporation [17].

The SHARPE program is useful both as an aid in learning about modeling [40] and as a tool for use in modeling real systems [2, 5, 11, 19, 43, 45, 46, 48]. Its applications have spanned computer systems dependability, network performance [6, 10, 23], wireless handoff [8, 26], aerospace reliability [53], space system reliability/availability [5], real time systems [1, 32, 18], railroad control systems, workflow systems [4], inspection-based preventive maintenance [50] and so on. It has been used to model reliability[3, 39, 38], performance [6, 10, 26, 30, 35, 38], availability [24, 58, 59], performability [21, 25], security[27], and survivability [13, 20]. In this paper, we will give some history of the work on SHARPE, discuss a couple of examples of its use, and talk about lessons learned.

2. SHARPE DEVELOPMENT

The original SHARPE model types were fault trees, reliability block diagrams, series-parallel acyclic directed graphs, and Markov and semi-Markov models. Since 1986, many different people have made enhancements to SHARPE, including new model types, solution algorithms and types of measures. These are summarized in Table 1. A book was published in 1996 on the theory and applications of SHARPE [42].

During the mid to late eighties, development work on HARP [9], SAVE [12], SHARPE and SPNP [7] were proceeding concurrently in our research group at Duke and hence there has been a natural overlap in the engines of these packages, although each is unique in its own way. One example is the algorithm for the steady state solution of Markov chains, written by Phil Chimento, which is shared by SAVE, SHARPE and SPNP [7]. Another is the numerical transient solver for Markov models, implemented by Jogesh Muppla in both SHARPE and SPNP [31].

In addition to shared development, SHARPE has been incorporated in other tools. For instance, it is one of the engines in Boeing's Integrated Reliability Analysis Package [36].

There are extensions of SHARPE other than those that became part of the SHARPE package: non-product-form queueing networks, response time distribution in such networks (both implemented by Varsha Mainkar in her PhD thesis [28]), connection to Mathematica for additional symbolic analysis, a phase type-fitting algorithm by Manish Malhotra [29], statistical routines by A. V. Ramesh, and many more.

In 2005, based on Boeing needs, a new algorithm for reliability graph analysis was added to SHARPE by Dazhi Wang as a part of his PhD thesis [53]. This is an efficient algorithm for reliability upper and lower bounds and hence enables the SHARPE user to solve very large models. It was used to carry out the reliability analysis of a Boeing 787 Current Return Network for the purpose of certification by the FAA. Boeing has submitted the algorithm for a patent [55].

In addition to new models and algorithms, we have increased the power of the specification language. The use of

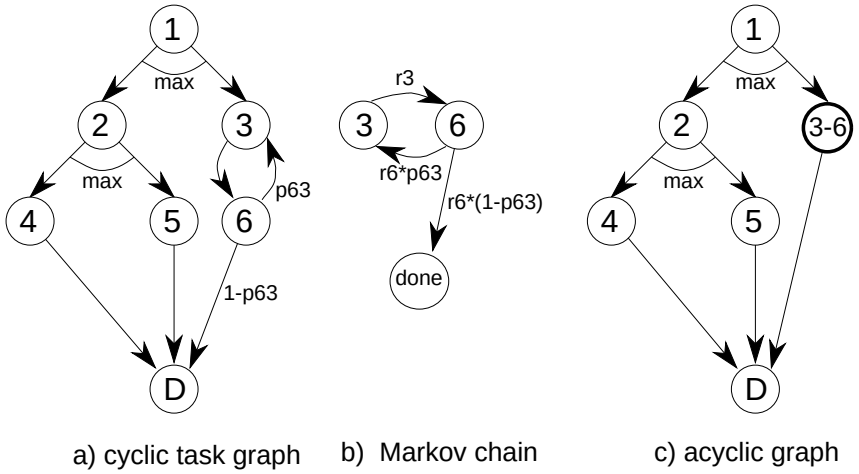

Figure 1: A task graph with a cycle and its exact decomposition.

SHARPE Enhancements	Authors	Year
closed single-chain product-form queueing network	Jogesh Muppala	1987
generalized stochastic Petri net (GSPN) model	Jogesh Muppala	1988
reliability graphs and repeated events in fault trees	Malathi Veeraraghavan [51]	1988
multichain PFQN	Varsha Mainkar	1991
improvement of fault tree algorithm	Tong Luo [22]	1993
semi-symbolic solver for phase-type Markov chains	A. V. Ramesh [37]	1995
algorithms for fault trees based on binary decision diagrams	Xinyu Zang [60]	1999
factoring algorithm for fault trees	Xinyu Zang	1999
multi-state fault trees	X. Zang [60]	1999
repeated edges in relgraphs	X. Zang [60]	1999
phased-mission systems	X. Zang [60]	1999
Markov regenerative processes	Wei Xie [57]	1999
fast MTTF algorithm for Markov chains and semi-Markov chains	Wei Xie [14, 57]	1999
stochastic Reward Nets	Hu Pan [33]	2001
steady-state MTTF computation for combinatorial, Markov and semi-Markov models based on binary decision diagrams	Dazhi Wang [52, 54]	2004
several non-exponomial distributions added to MRGP	Hiroyuki Okamura	2006

Table 1: SHARPE enhancements

a shorthand for specifying loops in Markov models, implemented by Robin Sahner, sometimes obviated the need to use GSPNs to express structured Markov models. In 2001, flex and bison (advanced versions of lex and yacc) were used to reconstruct the SHARPE parser in a backward compatible way [33]. Hu Pan extended the SHARPE language to allow for control statements such as "if" and "while" so that fixed-point iteration between sub-models can be carried out [26, 44].

When new algorithms were added to SHARPE, the existing algorithms were usually left in place. This sometimes lets users choose between multiple algorithms, and between getting an answer symbolic in the time variable or a numerical result for a particular value of t, or a mean. Sometimes it is useful to use results from different algorithms for the same model, or different kinds of model decomposition for the same problem, to cross-check the results.

When SHARPE was first written, we only had dumb terminals (and we walked 6 miles to school, uphill both ways). SHARPE read its model specification from a file and wrote results as text to standard output. It also had an interactive interface, printing prompts and accepting keyboard interface. This was useful for demonstrations, trying out very simple models, and providing a reminder of the syntax. Serious work was always done non-interactively, sometimes by writing a program to generate the text input for a particular model. Once graphical interfaces, and portable languages for writing them, became common, SHARPE was given a GUI. This was done in 2000 by Christophe Hirel [15].

3. EXACT MODEL DECOMPOSITION

When learning about modeling, SHARPE is useful in understanding the advantages of model decomposition. When modeling performance, if a task graph has no cycles and is series-parallel, the analysis of the model is straightforward and efficient. But if the graph has cycles or non-series-parallel pieces, the analysis is much more difficult.

It is possible to model the entire system with a Markov chain, but that makes the model much larger and more prone

```
markov inner
* transitions
3 6 r3
6 3 r6*p63
6 done r6*(1-p63)
end
* initial state
* probabilities
3 1
end
```

```
graph outer
* task precedence
1 2
1 3-6
2 4
2 5
4 9
5 9
3-6 9
end
```

```
* exit types
exit 1 max
exit 2 max
* time-to-completion
dist 1 exp(r1)
dist 2 exp(r2)
dist 4 exp(r4)
dist 5 exp(r5)
dist 9 exp(r9)
dist 3-6 cdf(inner)
end
```

Figure 2: Input for cyclic graph.

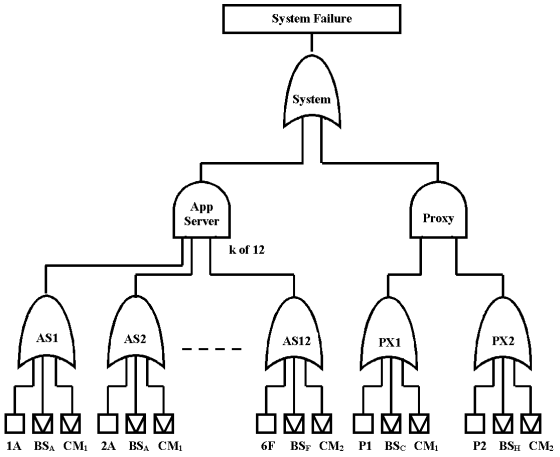

Figure 3: Top Level Fault Tree Model

to numerical error during analysis. A GSPN or a SRN-type Stochastic Petri Net model would be smaller, but these are often hard to construct properly, and it would not reduce the problem of numerical error. Instead, using SHARPE, we can model each cyclic or non-series-parallel piece with a Markov model (or a Petri net, or any other of the supported model types), and replace that set of tasks with one node representing the whole set.

Figure 1(a) (from [42]) shows an example of that. The time-to-completion distributions for the nodes are not shown; the distribution for each node n is the exponential distribution with parameter rn. We can isolate the cyclic part of the graph (tasks *3* and *6*) and model it using the Markov chain shown in Figure 1(b). We can replace the cyclic part of the graph in 1(a) by a single equivalent task whose time-to-completion distribution is the time-to-absorption distribution of the continuous-time Markov chain in Figure 1(b). The new upper-level graph is shown in Figure 1(c). This method yields an exact solution.

Figure 2 shows part of a file for input to SHARPE for this model. After the comment "time-to-completion", we assign a time-to-completion distribution function to each task in the upper-level model. For all tasks except the composite task *3-6* this is the same exponential distribution as for the original model. We assign to task *3-6* the distribution of the time to reach absorption in the Markov chain *inner*.

When all the time-to-finish distributions are exponential, it is possible to rewrite the entire graph as a Markov chain. This one is not too big to be analyzed, and provides a good consistency check. It also shows the value of having a tool for decomposing models. The complete Markov chain contains 16 states and 32 transitions. It took us over a half hour to set it up correctly by hand; the two-level model took less than five minutes to set up. The hierarchical model took about 0.3 seconds to solve as opposed to 1.9 seconds for the complete Markov chain.

4. EXPONENTIAL POLYNOMIALS

Where we assigned the exponential distribution above, we could have used any distribution that had exponential polynomial ("exponomial") form [42], being composed of sums of terms of the form $at^k e^{bt}$, where k is an integer and a and b are real. Examples are the exponential ($F(t) = 1-e^{-\lambda t}$), Erlang, hypoexponential and hyperexponential distributions.

Exponomials are easy for programs to represent and manipulate. They are closed under most of the operations used by the SHARPE model analysis algorithms: addition, multiplication, differentiation, integration, maximum, minimum, convolution, and probabilistic sum. The semi-numerical analysis ([37]) of Markov chains with cycles and absorbing states (phase-type chains) sometimes gives rise to functions with terms like $axe^{-t}\sin(t)$, but these can still be written as exponomials, if we allow complex values (occuring in conjugate pairs) for a and b. This is a remarkable feature of SHARPE, and provides large benefits to Markovian and non-Markovian analysis.

Not every distribution function of interest has exponomial form. An example commonly used in reliability modeling is the Weibull distribution,

$$W(t,b,c) = 1 - e^{-\frac{1}{b}t^c}.$$

Exponomials allow e^t but not e^{t^c}.

For non-state-space models, such as fault trees, SHARPE allows such distributions. Their presence precludes doing a semi-symbolic analysis, but SHARPE provides a numerical solution, providing probabilities for particular times t upon request. The Weibull distribution is one of many useful distributions available as built-in functions in SHARPE.

For state-space models, non-exponomial distributions can be approximated with exponomials [29]. If an exponomial corresponds to a phase-type distribution, we can insert it into a state-space model to obtain an approximate Markov model with an expanded state-space [47]. An alternative is to use either a semi-Markov model or a Markov regenerative process (MRGP) [8, 57, 59].

5. SHARPE APPLIED TO A REAL SYSTEM

SHARPE was used in the reliability and availability analysis of the SIP (Session Initiation Protocol) implemented on an IBM WebSphere [49]. It models a SIP service consisting of the WebSphere Application Server and a proxy server running on IBM BladeCenter hardware. This configuration provides high availability by using hardware and software redundancy and escalated levels of recovery. A multi-level SHARPE model can be used to predict the system availability. The top level is a fault tree (shown in Figure 3) whose nodes represent all of the software and hardware failures. At this level, we capture the system structure that tells us whether the whole system is available given the state of the software and hardware subsystems.

Some of the hardware subsystems can be broken down into

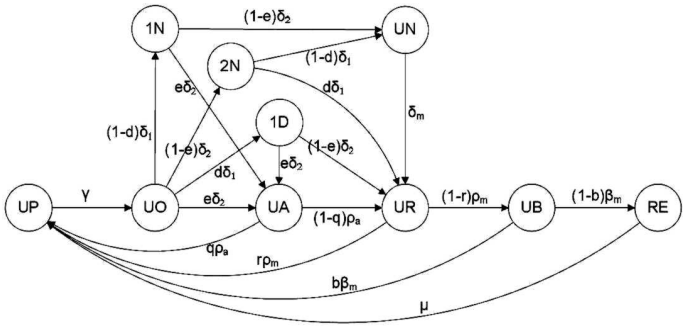

Figure 4: Availability model for application server

more detailed fault trees. In [49], these were included in the top-level fault tree model, but they could also have been separated into mid-level models. Which to choose depends on the nature of the system, the complexity of the models at each level, the modelers' preference, and the behavior of the analysis algorithms. One of the advantages of SHARPE is that it leaves these decisions up to the user.

Some subsystems represented by fault tree leaves [44] can have states where the system is up but with some components non-operational and undergoing repair. Failures are sometimes followed by successful reconfiguration ("covered") but sometimes bring the system down. Such systems are not easily modeled with fault tees, but can be modeled with Markov chains like the one in Figure 4. The SIP system was modeled with several different Markov chain submodels of fault tree leaves. Besides system availability, a user-perceived reliability measure called the defects per million (DPM) was also computed. This required us to model the interaction between call flow and the failure/recovery behavior. We were able to develop equations that could be eventually mapped into cumulative transient analysis of Markov reward models in SHARPE [53].

6. THE CHANGING WORLD OF SOFTWARE

When SHARPE was written, it was early in the world of programming languages and software engineering. It was after "go-to considered harmful" but well before object orientedness. It was before C++, Java and Perl, even before lex and yacc. The very first SHARPE versions were written in Pascal, but it was soon rewritten in C. It was the language that provided the power we needed, it was becoming standardized, and there were compilers available for many different kinds of hardware and operating systems.

We found C to be a fine language for our purposes, but it did present a problem in that it is prone to letting programmers create memory management problems. We found it relatively easy to find and fix problems with bad pointers. Memory leaks were a harder problem, usually showing up only in practice, and were hard to diagnose. Before long, we built some memory leak detection code into the software. Now there are static analysis tools to help with that. Many of the newer languages handle memory management internally, a clear advantage over C.

These days, there are a lot more options with regard to portability, but we have not been tempted to rewrite SHARPE using any of them. An interpreted language like Perl is more universally available but a lot slower in execution. A byte-compiled language like Java is somewhat more portable but can still be slower in execution. Furthermore, our experience has been that these languages often have more issues than a compiled language with differing behavior between different releases and even between different machines running the same release.

When object-oriented programming was introduced, we considered rewriting SHARPE in an O-O language. It is especially tempting to use Java, which we have found to be a friendly language, and have seen used quite successfully in a variety of applications. Some of the commercial O-O code we've read, especially C++, has shown that it's as easy to write bad code in an O-O language as in any other language. The code that does the real work is sometimes hard to find, and it is easy to make subtle mistakes that are very hard to diagnose. Furthermore, different people can have very different ideas of what an "object" should be for a particular problem domain. Having said that, we think there would be value in rewriting parts of SHARPE in a more explicitly object-oriented way, though we might want to do that while still using the C language.

Although the early days of SHARPE were way before there was much talk in most courses about the role of testing in software development, common sense and experience told us that any lines of code we had not exercised had a very good chance of being wrong. So we created a regression testbed - a set of tests to verify that the software works correctly. The testbed consists of examples of all the model types used in various combinations and the known good output for each example and a simple shell script to execute SHARPE for each input file and compare the output to the known good output. In anticipation of some aspects of "extreme programming" and "continuous integration", we used frequent test runs to help keep from introducing errors as we made fixes and enhancements to the program. Another advantage to having the testbed was that it gave us a way to assure backward compatibility as we introduced enhancements. Fairly early on, we started to use gcov to get coverage numbers for the regression tests, and we drove the branch coverage up to about 97%. Later, ATAC ([16]) was used to evaluate coverage.

7. NUMERICAL ISSUES IN SHARPE

There have been two key aspects to working on SHARPE. On the theoretical side, there is the study of the applicability and characteristics of methods of analyzing the models. On the practical side, there has been the need to implement the methods in a reliable and maintainable way. One of the challenges has been dealing with numerical error. The SHARPE program monitors itself for the buildup of numerical error, and notifies the user if it gets too big. If a particular model has its numerical error get out of hand, users can decompose their model into levels to work around the problem. Our set of regression tests includes examples that deliberately push the limits of the methods. We increased our confidence in the results by comparing the numbers from different analysis methods for the same model executed on the same hardware and comparing numbers from the same analysis method executed on different hardware.

8. CONCLUSION

SHARPE has been a long-lived modeling tool. We think

its usefulness results from the many kinds of model types offered, the flexibility offered by letting users choose how to combine the models, the relative ease of making enhancements, the user support (including the use of bugzilla [56]), and the ongoing high level of testing. It is in use at many universities as a teaching tool and as an aid in research in dependable computing. It has been used at many industrial sites to solve practical problems and has been integrated as an engine ([36, 34]) in other software packages. SHARPE can be acquired by sending an e-mail to kst@ee.duke.edu.

9. ACKNOWLEDGEMENT

We would like to thank Amita Devaraj, Rivalino Matias Jr, and Hiroyuki Okamura for help with this paper.

10. REFERENCES

[1] J. Aidemark, P. Folkesson, and J. Karlsson. Framework for node-level fault tolerance in distributed real-time systems. *DSN 2005*, 2005.

[2] A. Avritzer, A. Bondi, M. Grottke, K. Trivedi, and E. Weyuker. Performance assurance via software rejuvenation: monitoring, statistics and algorithms. In *Proc. International Conference on Dependable Systems and Networks*, 2006.

[3] J. B. Bowles and J. Gregory Dobbins. Approximate reliability and availability models for high availability and fault-tolerant systems with repair. In *Quality and Reliability Engineering International*, 2004.

[4] G. A. Chaparro-Baquero, N. G. Santiago, W. Rivera, and J. Fernando Vega-Riveros. Petri net workflow modeling for digital publishing measuring quantitative dependability attributes. In *Proc. IEEE Int. Symp. on Dependable Autonomic and Secure Computing*, 2006.

[5] D. Chen, S. Dharmaraja, D. Chen, K. Trivedi L. Li, R. Some, and A. Nikora. Reliability and availability analysis for the jpl remote exploration and experimentation system. In *Proc. International Conference on Dependable Systems and Networks*, 2002.

[6] I. Chen, T. Chen, and C. Lee. Agent-based forwarding strategies for reducing location management cost in mobile networks. *Mobile Network Applications*, 2001.

[7] G. Ciardo, A. Blakemore, P. F. Chimento, J. Muppala, and K. Trivedi. Automated generation and analysis of markov reward models using stochastic reward nets. In C. Meyer and R. Plemmons, editors, *Linear Algebra and Markov Chains and Queueing Models, IMA Volumes in Mathematics and its Applications*, volume 48. Springer-Verlag, 1992.

[8] S. Dharmaraja, K. Trivedi, and D. Logothetis. Performance modelling of wireless networks with generally distributed hand-off interarrival times. *Computer Communications Journal*, 2003.

[9] J. B. Dugan, K. Trivedi, M. Smotherman, and R. Geist. The Hybrid Automated Reliability Predictor. *AIAA J. Guidance, Control and Dynamics*, 1986.

[10] J. Gafsi and E. Biersack. Modeling and performance comparison of reliability strategies for distributed video servers. *IEEE Trans. Parallel Distributed System*, 2000.

[11] S. Garg, Y. Huang, C. Kintala, K. Trivedi, and S. Yajnik. Performance and reliability evaluation of passive replication schemes in application level fault tolerance. *Proc. International Symposium on Fault Tolerant Computing (FTCS)*, 1999.

[12] A. Goyal, W. C. Carter, E. de Souza e Silva, S. Lavenberg, and K. Trivedi. The system availability estimator. In *Proc. IEEE Int. Symp. on Fault-Tolerant Computing*, 1986.

[13] P. Heegaard and K. Trivedi. Survivability quantification of communication services. *IEEE Symp. on Dependable Systems and Networks*, 2008.

[14] P. Heidelberger, J. Muppala, and K. Trivedi. Accelerating mean time to failure computations. *Performance Evaluation*, 1996.

[15] C. Hirel, R. A. Sahner, X. Zang, and K. Trivedi. Reliability and performability modeling using sharpe 2000. *Computer Performance Evaluation / TOOLS*, 2000.

[16] J. R. Horgan and S. London. Atac: A data-flow coverage testing tool for c. In *Symp. on Assessment of Quality Software Development Tools*, 1992.

[17] O. Ibe, R. Howe, and K. Trivedi. Approximate availability analysis of vaxcluster systems. *IEEE Trans. on Reliability*, 1989.

[18] H. Kopetz, H. Kantz, G. Grunsteidl, P. Puschner, and J. Reisinger. Tolerating transient faults in mars. In *FTCS-20*, 1990.

[19] M. Lanus, L. Yin, and K. Trivedi. Hierarchical composition and aggregation of state-based availability and performability models. *IEEE Transactions on Reliability*, 2003.

[20] Y. Liu, V. Mendiratta, and K. Trivedi. Survivability analysis of telephone access network. In *Proc. Int. Symp. on Software Reliability Engineering*, 2004.

[21] N. Lopez-Benitez and K. Trivedi. Multiprocessor performability analysis. *IEEE Trans. on Reliability*, 1993.

[22] T. Luo and K. Trivedi. An improved algorithm for coherent system reliability. *IEEE Trans. on Reliability*, 1998.

[23] M. Lyu, X. Chen, and T. Wong. Design and evaluation of a fault-tolerant mobile-agent system. *IEEE Intelligent Systems*, 2004.

[24] M. Lyu and V. Mendiratta. Software fault tolerance in a clustered architecture: Techniques and reliability modeling. *Proc. of IEEE Aerospace Conference*, 1999.

[25] Y. Ma, J. Han, and K. Trivedi. Composite performance & availability analysis of wireless communication networks. *IEEE Trans. on Vehicular Technology*, 2001.

[26] B. Madan, S. Dharmaraja, and K. Trivedi. Combined guard channel and mobile-assisted handoff for cellular networks. *IEEE Trans. on Vehicular Technology*, 2008.

[27] B. Madan, K. Goseva-Popstojanova, K. Vaidyanathan, and K. Trivedi. Modeling and quantification of security attributes of software systems. *Performance Evaluation*, 2004.

[28] V. Mainkar. *Solutions of Large and Non-Markovian Performance Models*. PhD thesis, Duke University, 1994.

[29] M. Malhotra and A. Reibman. Selecting and implementing phase approximations for semi-markov models. *Stochastic Models*, 1993.

[30] M. Marsan, G. Conte, and G. Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor system. *ACM Trans. on Computer Systems*, 1984.

[31] J. Muppala, M. Malhotra, and K. Trivedi. Stiffness-tolerant methods for transient analysis of stiff markov chains. *Microelectronics and Reliability*, 1994.

[32] J. Muppala, K. Trivedi, and S. Woolet. Real-time-systems performance in the presence of failures. *IEEE Computer*, 1991.

[33] H. Pan. The reconstruction of sharpe. Master's thesis, Duke University, 2001.

[34] D. Raiteri, M. Iacono, G. Franceschinis, and V. Vittorini. Repairable fault tree for the automatic evaluation of repair policies. In *Proc. int. Conf. on Dependable Systems and Networks*, 2004.

[35] S. Ramani, K. Trivedi, and B. Dasarathy. Performance analysis of the corba notification service. *Proc. IEEE Symp. on Reliable Distributed Systems*, 2001.

[36] A. Ramesh, K. Trivedi, A. Somani, D. Twigg, U. Sandadi, and T. Sharma. An integrated reliability modeling environment. *Reliability Engineering and System Safety*, 1999.

[37] A. V. Ramesh and K. Trivedi. Semi-numerical transient analysis of markov models. *ACM Southeast Regional Conference*, 1995.

[38] R. Sahner and K. Trivedi. Performance and reliability analysis using directed acyclic graphs. *IEEE Trans. on Software Engineering*, 1987.

[39] R. Sahner and K. Trivedi. Reliability modeling using sharpe. *IEEE Trans. on Reliability*, 1987.

[40] R. Sahner and K. Trivedi. A software tool for learning about stochastic models. *IEEE Transactions on Education*, 1993.

[41] R. A. Sahner. *A hybrid, combinatorial-Markov method of solving performance and reliability models*. PhD thesis, Duke University, 1986.

[42] R. A. Sahner, K. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers, 1996.

[43] N. Sato, H. Nakamura, and K. Trivedi. Detecting performance and reliability bottlenecks of composite web services. *Proc. ICSOC*, 2007.

[44] W. Smith, K. Trivedi, L. Tomek, and J. Ackaret. Availability analysis of blade server systems. *IBM Systems Journal*, 2008.

[45] L. Tomek and K. Trivedi. Fixed-point iteration in availability modeling. In *Proceedings of the 5th International GI/ITG/GMA Conference on Fault-Tolerant Computing Systems, Tests, Diagnosis, Fault Treatment*, 1991.

[46] K. Trivedi. Availability analysis of cisco gsr 12000 and juniper m20/m40. *Cisco Technical Report*, 2000.

[47] K. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley, second edition, 2001.

[48] K. Trivedi, R. Vasireddy, D. Trindade, S. Nathan, and R. Castro. Modeling high availability systems. *Proc. IEEE Pacific Rim International Symposium on Dependable Computing*, 2006.

[49] K. Trivedi, D. Wang, D. Hunt, A. Rindos, W. Smith, and B. Vashaw. Availability modeling of sip protocol on ibm websphere. *PRDC*, 2008.

[50] K. Vaidyanathan, D. Selvamuthu, and K. Trivedi. Analysis of inspection-based preventive maintenance in operational software systems. *Intl. Symp. on Reliable Distributed Systems*, 2002.

[51] M. Veeraraghavan and K. Trivedi. An improved algorithm for symbolic reliability analysis. *IEEE Transactions on Reliability*, 1991.

[52] D. Wang. Mttf computation for analytical models. Master's thesis, Duke University, 2004.

[53] D. Wang. *Service Reliability: Models, Algorithms and Applications*. PhD thesis, Duke University, 2007.

[54] D. Wang and K. Trivedi. Computing steady-state mean time to failure for non-coherent repairable systems. *IEEE Transactions on Reliability*, 2005.

[55] D. Wang, K. Trivedi, T. Sharma, A. Ramesh, D. Twigg, L. Nguyen, and Y. Liu. A new reliability estimation method for large systems. *The Boeing Company patent application pending*, 2008.

[56] www.borel.ee.duke.edu/bugzilla.

[57] W. Xie. Markov regenerative process in sharpe. Master's thesis, Duke University, 1999.

[58] W. Xie, Y. Cao, H. Sun, and K. Trivedi. Modeling of user perceived webserver availability. *IEEE Int. Conf. on Communications*, 2003.

[59] L. Yin, R. Fricks, and K. Trivedi. Application of semi-markov process and ctmc to evaluation of ups system availability. *Proc. Reliability and Maintainability Symposium*, 2002.

[60] X. Zang. *Dependability Modeling of Computer Systems and Networks*. PhD thesis, Duke University, 1999.