Chapter 22
Reliability and Availability Analysis in Practice

Kishor Trivedi and Andrea Bobbio

Abstract Reliability and availability are key attributes of technical systems. Methods of quantifying these attributes are thus essential during all phases of system lifecycle. Data (measurement)-driven methods are suitable for components or subsystems but, for the system as a whole, model-driven methods are more desirable. Simulative solution or analytic–numeric solution of the models are two major alternatives for the model-driven approach. In this chapter, we explore model-driven methods with analytic–numeric solution. Non-state-space, state-space, hierarchical, and fixed-point iterative methods are explored using real-world examples. Challenges faced by such modeling endeavors and potential solutions are described. Software package SHARPE is used for such modeling exercises.

Keywords Availability · Reliability · Fault tree · Markov model · Non-state-space model · State-space model · Hierarchical model · Fixed-point iteration technique

22.1 Introduction

This chapter discusses techniques that are found to be effective for reliability and availability assessment of real systems. Modern life heavily relies on man-made systems that are expected to be reliable. Many high-tech cybersystems are found wanting since their failures are not so uncommon. Such failures and consequent downtimes lead to economic losses, to a loss of reputation, and to even loss of lives. To ameliorate the situation, methods have been developed that reduce failure occurrences and resultant downtimes. In order to gauge the effectiveness of these improvement methods, scalable and high-fidelity techniques of reliability and availability assessment are needed.

K. Trivedi (✉)
Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA
e-mail: ktrivedi@duke.edu

A. Bobbio
DiSit, Università del Piemonte Orientale, 15131 Alessandria, Italy

K. B. Misra (ed.), *Handbook of Advanced Performability Engineering*,
https://doi.org/10.1007/978-3-030-55732-4_22

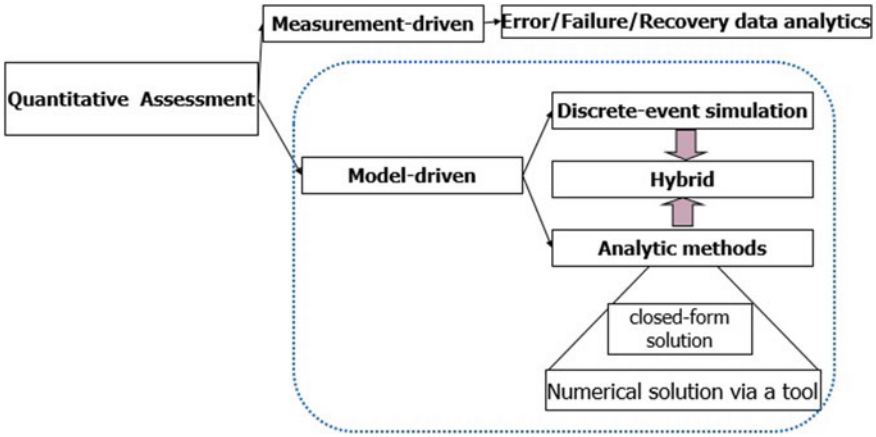

Fig. 22.1 Reliability/availability assessment methods

This chapter discusses techniques that are found to be effective for reliability and availability assessment in practice. Assessment methods can be divided into measurement-driven (or data-driven) versus model-driven methods (Fig. 22.1). Data-driven methods are suitable for small subsystems, while model-driven methods are appropriate for large systems. Using model-driven methods, we can derive the dynamic behavior of a system consisting of many components using first principles (of probability theory) rather than from measurements.

In practice, these two approaches are combined together so that subsystem or component behavior is derived using data-driven methods, while the system behavior is derived using model-driven methods.

This chapter focuses on model-driven methods. Models can be solved using discrete-event simulation or using analytic–numeric techniques. Some simple models can be solved analytically to yield a closed-form formula while a much larger set of models can be dealt with by a numerical solution of their underlying equations. The latter approach is known as analytic–numeric solution. Distinction between analytic–numeric solution versus discrete-event simulation-based solution ought to be noted. We believe that simulative solution and analytic–numeric solutions should be judiciously combined in order to solve complex system models. This chapter, however, is on analytic–numeric methods, providing an overview of a recently published book by the authors of this chapter [1].

Our approach to exposing the methods is example-based. Chosen examples are of real systems that we have ourselves analyzed for some companies. Overall modeling process is depicted in Fig. 22.2.

Non-state-space (or combinatorial) models can deal with large systems if based on the drastic assumption of statistical independence among components. State-space model types, specifically continuous-time Markov chains and Markov reward models, are commonly utilized for higher fidelity. Multi-level models that judiciously combine non-state-space and state-space methods will be seen to have the scalability

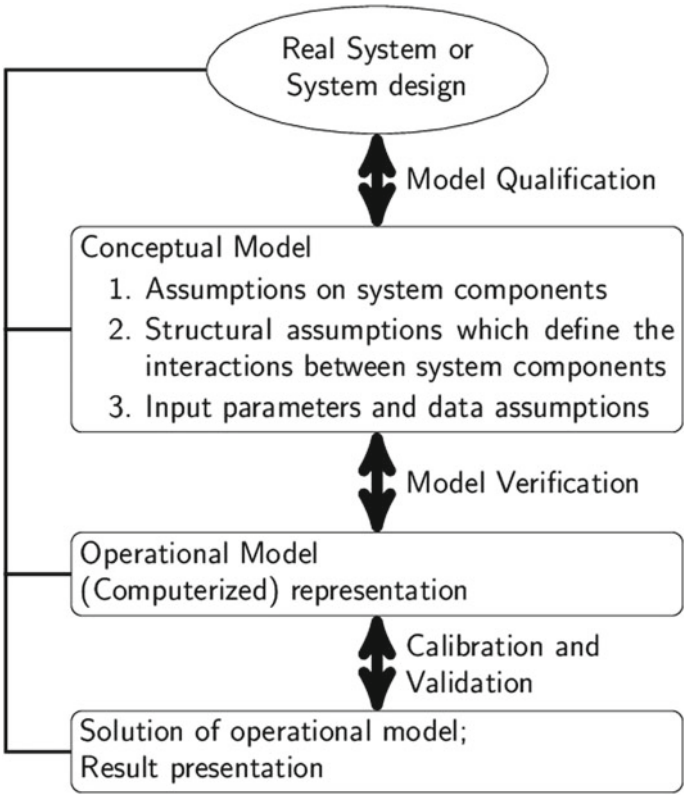

Fig. 22.2 The overall modeling process

and fidelity needed for capturing the dynamic behavior of real systems. Depending on the application, a model may be solved for its long-term (steady-state) behavior or its time-dependent (or transient) behavior. Solution types for such models are classified in Fig. 22.3 [1, 2]. Software packages that are used in solving the examples of this chapter are SHARPE [2, 3] and SPNP [4, 5].

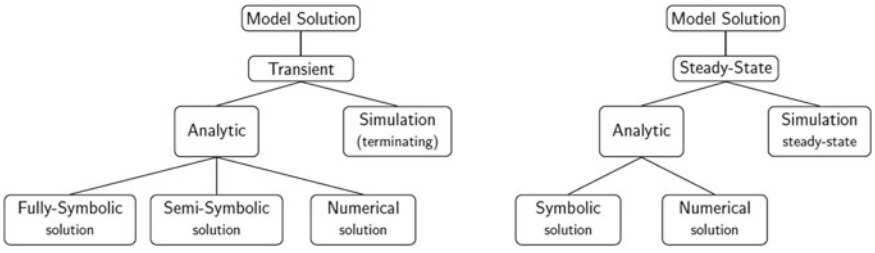

Fig. 22.3 Solution techniques

22.2 Non-state-Space Methods

Several traditional methods for the analysis of system reliability and availability can be classified under the umbrella of non-state-space (sometimes called combinatorial) methods:

- Reliability Block Diagrams (RBD)
- Network reliability or Reliability graphs (RelGraph)
- Fault Trees.

The simplest paradigm for reliability/availability is the (series–parallel) reliability block diagram (RBD). These are commonly used in computer and communications industry and are easy to use and assuming statistical independence, simple algorithms are available to solve very large RBDs. Reliabilities (availabilities) multiply for blocks in series, while unreliabilities (unavailabilities) multiply for blocks in parallel. Efficient algorithms for *k-out-of-n* blocks are also available, both in the case of statistically identical blocks and for non-identical blocks [1].

Besides system reliability at time t, system mean time to failure, system availability (steady-state and instantaneous), and importance measures can also be computed so as to point out critical components (bottlenecks) [1].

High availability requirement in telecommunication systems is usually more stringent than most other sectors of industry. The carrier-grade platform from Sun Microsystems requires a "five nines and better" availability. From the availability point of view, the top-level architecture of a typical carrier-grade platform was modeled in [6] as a reliability block diagram consisting of series, parallel, and *k-out-of-n* subsystems, as shown in Fig. 22.4. The SCSI series block is further expanded as in the inset of Fig. 22.4.

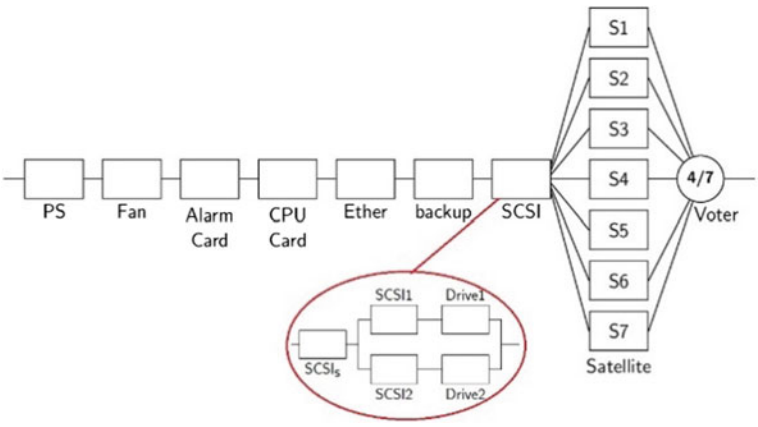

Fig. 22.4 High availability platform from sun microsystems

Series–parallel structure is often violated in practice. Non-series–parallel block diagrams are often cast as s-t connectedness problems, also known as network reliability problems or just relgraph in SHARPE. The price to be paid for this additional modeling power is the increased complexity of solution methods. Known solution methods are factoring (or conditioning), finding all minpaths followed by the use of one of many sum-of-disjoint-product (SDP) algorithms, the use of binary decision diagrams (BDD), or the use of Monte Carlo simulation. SDP- and BDD-based algorithms have been implemented in the SHARPE software package [2, 3]. Nevertheless, real systems pose a challenge to these algorithms. For instance, the reliability of the current return network subsystem of Boeing 787 was modeled as a relgraph shown in Fig. 22.5. However, the number of minpaths was estimated to be over 4.2 trillion.

To solve the model, for the purpose of FAA certification, a new bounding algorithm was developed, patented, and published [7]. Table 22.1 reports the results showing that the upper and lower bounds to the s-t reliability were close enough, with a very small number of minpaths and mincuts selected for the computation. The computation time was very short for this otherwise intractable problem. This new bounding algorithm is implemented in the SHARPE software package and continues to be

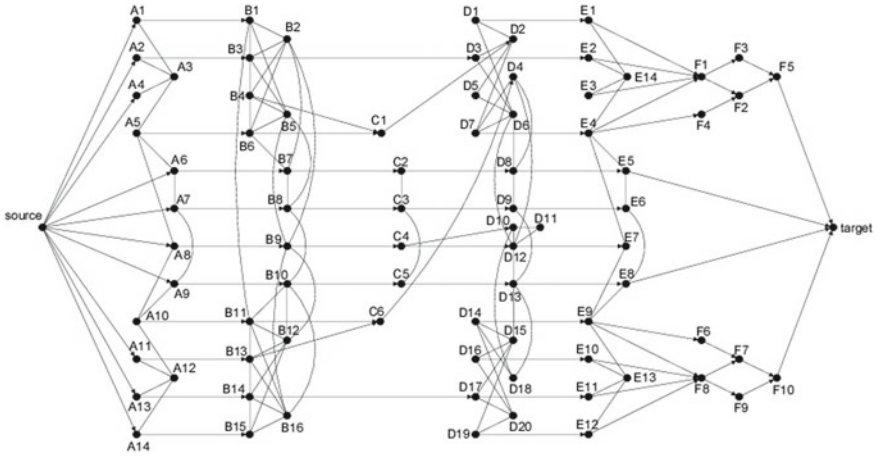

Fig. 22.5 Boeing relgraph example

Table 22.1 Unreliability upper/lower bounds

Runtime	20 s	120 s	900 s
Up bound	1.146036×10^{-8}	1.081432×10^{-8}	1.025519×10^{-8}
Low bound	1.019995×10^{-8}	1.019995×10^{-8}	1.019995×10^{-8}
#minpaths	28	29	33
#mincuts	113	113	113

used by Boeing (via their IRAP software package [8]) for the reliability assessment of current return network of all Boeing commercial airplanes.

Table 22.2 shows a comparison of SDP and BDD methods for various benchmark networks of increasing complexity. The different BDD columns show the effect of node ordering on the computational time [9]. The used benchmark networks are shown in Fig. 22.6 and were inspired by the literature [10]. Note also that the bounding method is not utilized in the comparison table.

In the aerospace, chemical, and nuclear industries, engineers use fault trees (FT) to capture the conditions under which system fails. These Boolean conditions are encoded into a tree with AND gates, OR gates, and *k-out-of-n* gates as internal nodes, while leaf nodes represent component failures and the top or root node indicates system failure.

Fault trees without repeated events are equivalent to series–parallel RBDs, while those with repeated events are more powerful [1, 2, 11]. Solution techniques for fault trees with repeated events are the same as those for the network reliability problem discussed in the previous paragraph [1]. Fault trees with several thousand components can be solved with relative ease.

Figure 22.7 shows an FT for a GE Equipment Ventilation System. Notice that leaves drawn as circles are basic events, while inverted triangles represent repeated events. Assuming that all the events have a failure probability equal to $q = 0.001$, the SHARPE input file and the SHARPE output file are shown in Fig. 22.8 on the left-hand and on the right-hand side, respectively. In this example, SHARPE is asked to compute the Top Event probability (QTE = 1.0945e−02) as well as the list of the mincuts. We could ask for importance measures as well as a closed-form expression of top event probability [1, 3]. By assigning failure rates for each event, we could ask for the time-dependent failure probability of the system. Many other possibilities for output measures exist.

By assigning failure rates to components, system reliability at time t and the mean time to system failure can be computed. Time-to-failure distribution other than exponential (e.g., Weibull) can be used in such non-state-space models. Furthermore, by assigning failure rate and repair rate to each component, steady-state and instantaneous availability can be computed (assuming independence in repair besides failure independence).

FTs have been extended to non-coherent gates such as NOT gates, to multi-state components [12], phased-mission systems [13], and with dynamic gates [14]. SHARPE fault trees allow NOT gate, multi-state components, and phased-mission systems. Dynamic gates are not explicitly included in SHARPE but can easily be implemented since (static) fault trees, Markov chains, and their combination via hierarchical modeling are provided [1].

Table 22.2 Comparison of SDP and BDD with various orderings

Example	SDP		BDD(O1)		BDD(O2)		BDD(O3)		BDD(O4)	
	#DP	Time(s)	Size	Time(s)	Size	Time(s)	Size	Time(s)	Size	Time(s)
Relex1	7	0.03	15	0.04	15	0.04	19	0.04	17	0.04
Relex2	11	0.04	27	0.05	19	0.05	27	0.05	27	0.05
Relex3	16	0.04	21	0.05	28	0.05	32	0.05	28	0.05
Relex4	40	0.05	28	0.05	57	0.05	54	0.05	33	0.05
Relex5	78	0.06	64	0.06	65	0.06	85	0.06	67	0.06
Relex6	150	0.10	291	0.08	347	0.08	277	0.08	277	0.08
Relex7	402	0.31	120	0.06	187	0.07	1178	0.12	444	0.09
Relex8	2294	6.46	966	0.16	505	0.19	4865	0.38	743	0.15
Relex9	47,312	148.45	2083	0.41	14,821	2.26	12,277	1.25	3360	0.46

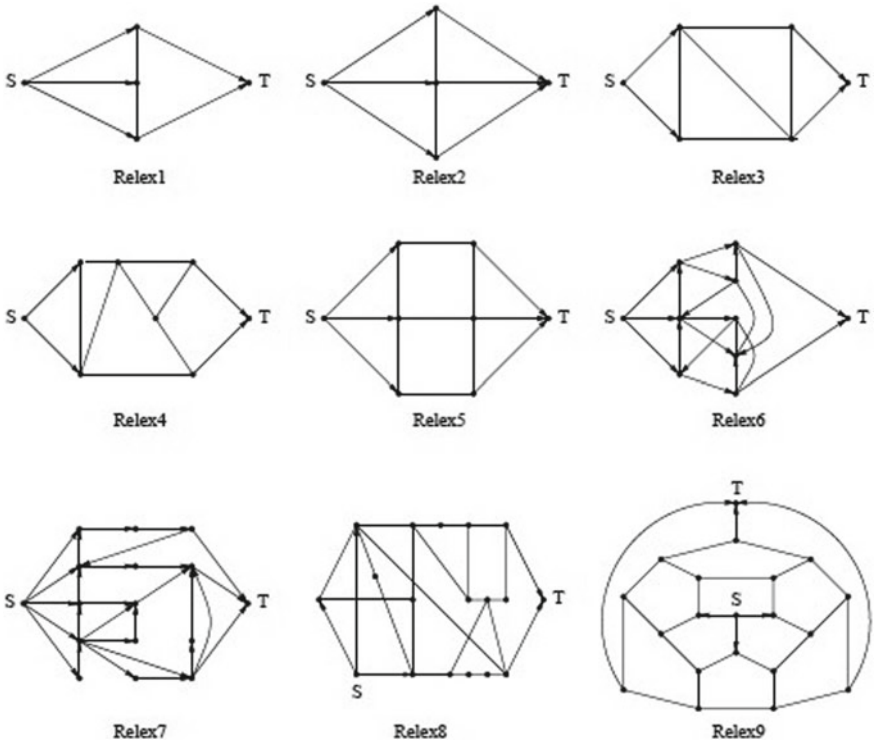

Fig. 22.6 Benchmark networks

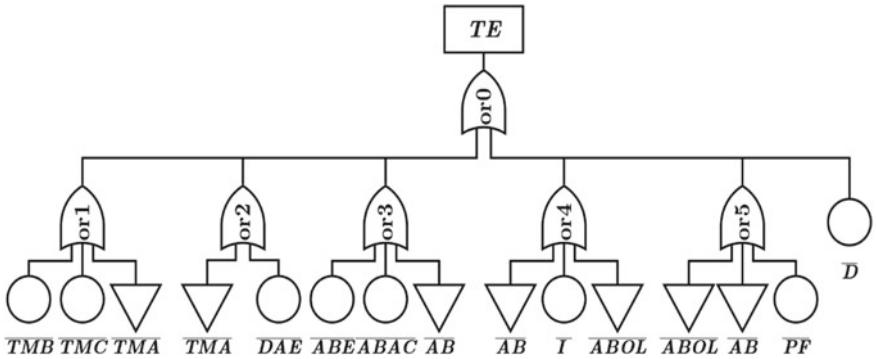

Fig. 22.7 Fault tree model equipment ventilation system

```
echo Ventilation System

ftree vent
basic TMB prob(lp)
basic TMC prob(lp)
repeat TMA prob(lp)
basic DAE prob(lp)
basic ABE prob(lp)
basic ABAC prob(lp)
repeat AB prob(lp)
basic I prob(lp)
basic PF prob(lp)
basic D prob(lp)
repeat ABOL prob(lp)
or orr1 TMB TMC TMA
or orr2 TMA DAE
or orr3 ABE ABAC AB
or orr4 AB I ABOL
or orr5 AB PF ABOL
or te orr1 orr2 orr3 orr4 orr5 D
end
bind
lp 0.001
end
var sysunrel sysprob(vent)
expr sysunrel

mincuts(vent)
end
```

```
Ventilation System
sysunrel:   1.0945e-002

Mincuts for system vent:

[(TMC)],
[(TMB)],
[(PF)],
[(ABOL)],
[(I)],
[(AB)],
[(ABAC)],
[(ABE)],
[(TMA))],
[(DAE)],
[(D)]
```

Fig. 22.8 SHARPE input/output files for ventilation system

22.3 State-Space Methods

As stated in the last section, non-state-space models with thousands of components can be solved without generating their underlying state space by making the independence assumption. But in practice, dependencies do exist among components. We then need to resort to state-space models such as (homogeneous) continuous-time Markov chains (CTMC).

Markov models have been used to capture dynamic redundancy, imperfect coverage (e.g., failure to failover or failure to detect, etc.), escalated levels

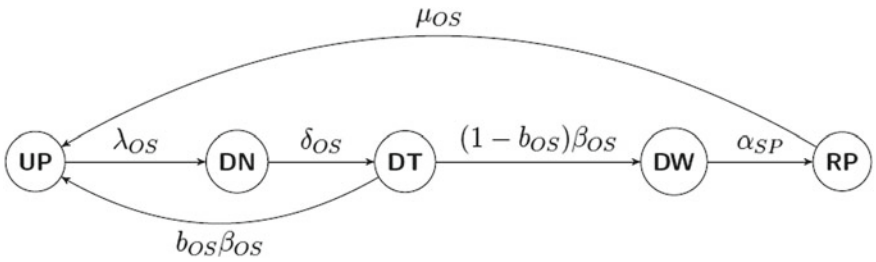

Fig. 22.9 CTMC availability model of Linux OS

of recovery, concurrency, contention for resources, combined performance and reliability/availability, and survivability [1, 15]. Markov availability model will have no absorbing states (Fig. 22.9), while Markov reliability models will have one or more absorbing states (Fig. 22.11). Markov models can be solved for steady-state, transient, and cumulative transient behavior according to the following equations [1, 15]:

Steady-state	$\boldsymbol{\pi}\,\boldsymbol{Q} = 0$ with $\sum \pi = 1$
Transient	$\mathrm{d}\boldsymbol{\pi}(t)/\mathrm{d}t = \boldsymbol{\pi}(t)\,\boldsymbol{Q}$ given $\boldsymbol{\pi}(0)$
Cumulative transient	$\mathrm{d}\boldsymbol{b}(t)/\mathrm{d}t = \boldsymbol{b}(t)\;\boldsymbol{Q} + \boldsymbol{\pi}(0)$

In the above formulas, $\boldsymbol{Q}$ is the infinitesimal generator matrix of the CTMC, $\boldsymbol{\pi}(t)$ is the state probability vector at time t, $\boldsymbol{\pi}(0)$ is the initial state probability vector, $\boldsymbol{\pi} = \lim_{t\to\infty} \boldsymbol{\pi}(t)$ is the steady-state probability vector, and $\boldsymbol{b}(t) = \int_0^t \boldsymbol{\pi}(u)\mathrm{d}u$ is the vector of the expected state occupancy times in the interval from 0 to t. Derivatives of these measures with respect to the input parameters can also be computed numerically [1].

22.3.1 CTMC Availability Models

The system availability (or instantaneous, point, or transient availability) is defined as the probability that at time t the system is in an up state:

$$A(t) = P\{\text{system working at } t\}$$

Steady-state availability (A_{ss}) or just availability is the long-term probability that the system is up:

$$A_{\text{ss}} = \lim_{t\to\infty} A(t) = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

where MTTF is the system mean time to failure and MTTR is the system mean time to recovery. When applied to a single component, the above equation holds without any distributional assumptions. For a complex system with redundancy, the equation holds if we use "equivalent" MTTF and "equivalent" MTTR [1].

The availability model of the Linux operating system used in IBM's SIP implementation on WebSphere was presented in [16] and is shown in Fig. 22.9. From the up state, the model enters the down state DN with failure rate λ_{OS}. After failure detection, with a mean time of $1/\delta_{OS}$, the system enters the failure-detected state DT.

The OS is then rebooted with the mean time to reboot given by $1/\beta_{OS}$. With probability b_{OS} the reboot is successful, and system returns to the UP state. However, with probability $1 - b_{\text{OS}}$, the reboot is unsuccessful, and the system enters the DW state where a repairperson is summoned. The travel time of the repairperson is assumed to be exponentially distributed with rate α_{SP}. The system then moves to the state RP. The repair takes a mean time of $1/\mu_{OS}$, and after its completion, the system returns to the UP state.

Solving the steady-state balance equations, a closed-form solution for the steady-state availability of the OS is easily obtained in this case due to the simplicity of the Markov chain.

$$A_{\text{ss}} = \pi_{\text{UP}} = \frac{1}{\lambda_{\text{OS}}}\left[\frac{1}{\lambda_{\text{OS}}} + \frac{1}{\delta_{\text{OS}}} + \frac{1}{\beta_{\text{OS}}} + (1 - b_{OS})\left(\frac{1}{\alpha_{SP}} + \frac{1}{\mu_{OS}}\right)\right]^{-1}$$

We can alternatively obtain a numerical solution of the underlying equations by using a software package such as SHARPE. Either graphical or textual input can be employed. The SHARPE textual input file modeling the CTMC of Fig. 22.9 is shown in Fig. 22.10. Noting that UP (labeled 1) is the only upstate, the steady-state availability is computed using the command *expr prob (LinuxOS,1)*. With the assigned numerical values for parameters (see Fig. 22.10), the result is $A_{\text{ss}} = 0.99963$.

```
echo Linux OS in IBM WebSphere Model
markov LinuxOS
1 2 los
2 3 dos
3 1 bos*beta
3 4 (1-bos)*beta
4 5 asp
5 1 mos

 Parameter values
```

```
bind
los 1/4000
dos 1
beta 6
bos 0.9
asp 1/2
mos 1
end
echo  Steady-state availability equal
echo  to probability of state 1
expr prob (LinuxOS,1)
end
```

Fig. 22.10 SHARPE input file for the CTMC of Fig. 22.9

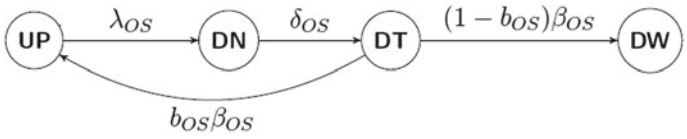

Fig. 22.11 CTMC reliability model of Linux OS

22.3.2 CTMC Reliability Models

While CTMC availability models have no absorbing states, CTMC reliability models have one or more absorbing states and the reliability at time t is defined as the probability that the system is continuously working during the interval $(0 - t]$. Further, since in a reliability model the system down state is an absorbing state, the MTTF can be calculated as the mean time to absorption in the corresponding CTMC model [1, 2, 15].

The reliability model extracted from the availability model of the Linux operating system used for IBM's SIP application is shown in Fig. 22.11. The repair transition from state RP to state UP and the transition from state DW are removed, that is, the down state reached starting from the UP state is made an absorbing state. Note that states DN and DT are down states but the sojourns in these states are likely to be short enough to be considered as glitches that can be ignored while computing system reliability and MTTF.

In this case, the model is simple enough so that a closed-form solution can be obtained by hand (or using Mathematica) by setting up and solving the underlying Kolmogorov differential equations. Alternatively, a numerical solution of the underlying equations can be obtained using SHARPE. The SHARPE textual input file for the reliability model of Fig. 22.11 is shown in Fig. 22.12. Note that in this case, since the CTMC is not irreducible, an initial probability vector must be specified.

The system reliability at time t is defined in this case as $R(t) = \pi_{\mathrm{UP}}(t)$ and, in the SHARPE input file of Fig. 22.11, is computed from $t = 0$ to $t = 10{,}000$ in steps of 2000. As noted earlier, the MTTF is defined as the mean time to absorption and is computed using the SHARPE command *expr mean (LinuxOS)*. With the assigned numerical values, the result is MTTF = 40,012 h.

The CTMC of a reliability model can be, but need not be, acyclic, as in the case of Fig. 22.11. If there is no component level repair (recovery), then the CTMC will be acyclic but if there is component level repair (but no repair after system failure) then the CTMC will have cycles. However, the model will always have one or more absorbing states.

Reliability modeling techniques have wide applications in different technical fields and have been proposed to provide new frontiers in predicting healthcare outcomes. With the rise in quantifiable approaches to health care, lessons from reliability modeling may well provide new ways of improving patient healthcare. Describing the development of conditions leading to organ system failure provides motivation for quantifying disease progression. As an example, a simple model for

```
markov LinuxOS
1 2 los
2 3 dos
3 1 bos*beta
3 4 (1-bos)*beta
end

* initial state probabilities
1 1.0
2 0.0
3 0.0
4 0.0
end

* Parameter values
bind
los 1/4000
dos 1
beta 6
bos 0.9
end
```

```
echo  Reliability vs time equal to probability
echo  of state 1  at time t

echo System reliability at times 0 thru 10000
echo in steps of 2000
func rel(t) tvalue(t;LinuxOS,1)
loop t,0,  10000, 2000
expr rel(t)
end
expr mean(LinuxOS)
end
```

Fig. 22.12 SHARPE input file for CTMC of Fig. 22.11

progressive kidney disease leading to renal failure is reported in Fig. 22.13 [17] where five discrete conditions are enumerated in keeping with clinical classification of kidney function.

The parameter values, used in solving the model of Fig. 22.13, are reported in Table 22.3. These values are estimated for a 65-year-old Medicare patient and are

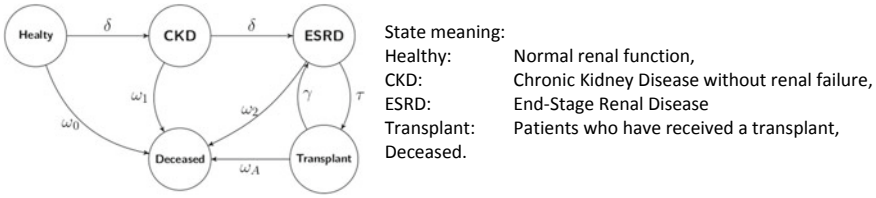

Fig. 22.13 Markov model of renal disease

Table 22.3 Parameter values for a 65-year-old medicare patient

Description	Symbol	Value (event/year)
Decline	δ	0.1887
Transplant	τ	0.1786
Graft rejection	γ	0.0050
Prognosis-healthy	ω_0	0.0645
Prognosis-CKD	ω_1	0.1013
Prognosis-ESRD	ω_2	0.2174
Prognosis-Transplant	ω_A	0.0775

based on the latest available statistics from the United States Renal Data System (USRDS) annual report [18].

The model of Fig. 22.13 is solved for the survival rate and expected cost incurred by a patient in a 1-year interval [17].

Efficient algorithms are available for solving Markov chains with several million states [19–21] both in the steady-state and in the transient regime. Furthermore, measures of interest such as reliability, availability, performability, survivability, etc. can be computed by means of reward rate assignments to the states of the CTMC [1, 15]. Derivatives (sensitivity functions) of the measures of interest with respect to input parameters can also be computed to help detect bottlenecks [22–24]. Nevertheless, the generation, storage, and solution of real-life-system Markov models still pose challenges. Higher level formalisms such as those based on stochastic Petri nets (SPNs) and their variants [4, 15, 25–27] have been used to automate the generation, storage, and the solution of large state-space Markov models [26]. Our own version of SPN is known as stochastic reward nets (SRN). SRNs extend SPN formalism in several useful ways besides allowing specification of reward rates at the net level. This enables more concise description of real-world problems and an easier way to get the output measures [4].

An example of the use of stochastic reward nets to model the availability of an Infrastructure-as-a-Service (IaaS) cloud is shown in Fig. 22.14 [28]. To reduce power usage costs, physical machines (PMs) are divided into three pools: Hot pool (high performance and high power usage), warm pool (medium performance and medium power usage), and cold pool (lowest performance and lowest power usage). PMs may fail and get repaired. A minimum number of operational hot PMs are required for the system to function but PMs in other pools may temporarily be assigned to the hot pool in order to maintain system operation. Upon repair, PMs migrate back to their original pool. Migration creates dependencies among the pools.

A monolithic CTMC is too large to construct by hand. We use our high-level formalism of Stochastic Reward Net (SRN) [26]. An SRN model can be automatically converted into an underlying Markov (reward) model that is solved numerically for the measures of interest such as expected downtime, steady-state availability, reliability, power consumption, performability, and sensitivities of these measures.

In Fig. 22.14, place P_{h} initially contains n_{h} tokens (PMs of the hot pool), P_{w} contains n_{w} tokens (PMs of the warm pool), and P_{c} contains n_{c} tokens (PMs of the cold pool). Assuming the number of PMs in each pool is identical and equal to n, the number of states for the monolithic model of Fig. 22.14, is reported in the second column of Table 22.4. From this table, it is clear that this approach based merely on a higher formalism such as SRN, which we call largeness tolerance, soon reaches its limits as the time needed for the generation and storage of the state space becomes prohibitively large for real systems.

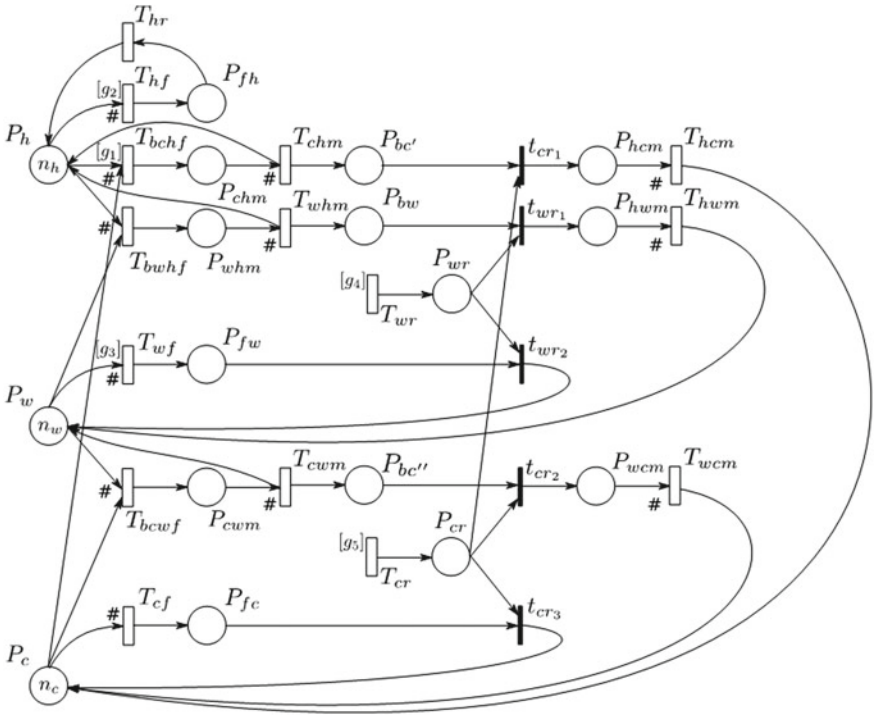

Fig. 22.14 SRN availability model of IaaS cloud

Table 22.4 Comparison of monolithic versus decomposed model

n	Monolithic model	Interacting sub-model
	#states	#states
3	10,272	196
4	67,075	491
5	334,948	1,100
6	1,371,436	2,262
7	4,816,252	3,770
8	Memory overflow	6,939
10	–	20,460
20	–	21,273
40	–	271,543
60	–	1,270,813
80	–	3,859,083
100	–	9,196,353

n is the initial #PM in each pool

22.4 Hierarchy and Fixed-Point Iteration

In order to avoid large models as is the case in a monolithic Markov (or generally state space) model, we advocate the use of multi-level models in which the modeling power of state-space models and efficiency of non-state-space models are combined together (Fig. 22.15).

Since a single monolithic model is never generated and stored in this approach, this is largeness avoidance in contrast with the use of largeness tolerance (recall stochastic Petri nets, SRNs, and related modeling paradigms) wherein the underlying large model is generated and stored. In multi-level modeling, each of the models is solved and results are conveyed to other relevant models to use as their input parameters. This transmission of results of one sub-model as input parameters to other sub-models is depicted as a graph that we have called an import graph [29].

Consider, for instance, the availability model of the SUN Microsystem whose top-level RBD availability model is shown in Fig. 22.4. Each block of the RBD of Fig. 22.4 is a complex subsystem that was modeled separately using the appropriate formalism in order to compute the steady-state availability of that subsystem. In the present case, the subsystems were modeled as Markov chains to cater to the dependencies within each subsystem.

The subsystem availability is then rolled up to the higher level RBD model to compute the system steady-state availability. The import graph for this system model is shown in Fig. 22.16. Specification, solution, and passing parameters for such multi-level models are facilitated by the SHARPE software package [2, 3]. The import graph in this case is acyclic. We can then carry out a topological sort of the graph resulting in a linear order specifying the order in which the sub-models are to be solved and the results rolled up in the hierarchy.

As the next example, we return to the IaaS cloud availability model and improve its scalability. The monolithic SRN model of Fig. 22.14 is decomposed into three sub-models to describe separately the behavior of the three pools [28, 29] while taking into account their mutual dependencies by means of parameter passing. The three sub-models are shown in Fig. 22.17.

Its import graph is shown in Fig. 22.18, indicating the output measures and input parameters that are exchanged among sub-models to obtain the overall model solution. Import graphs such as the one shown in Fig. 22.18 are not acyclic, and hence the

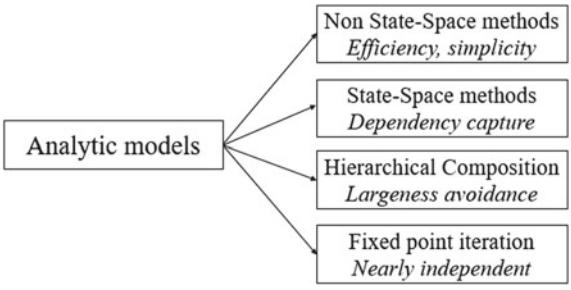

Fig. 22.15 Analytic modeling taxonomy

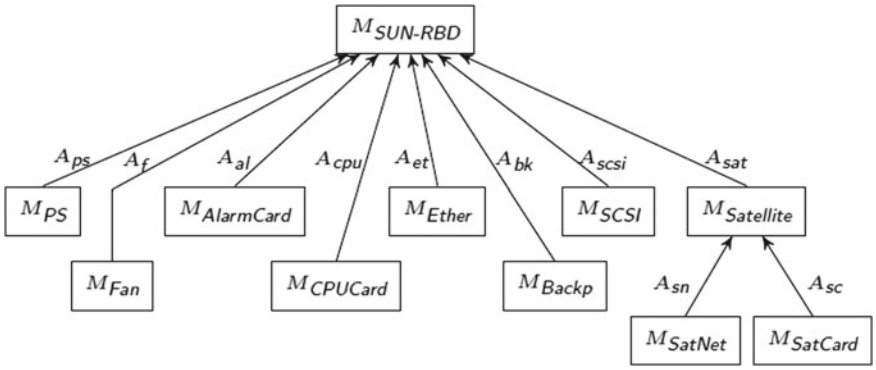

Fig. 22.16 Import graph for high availability platform from Sun Microsystems [6]

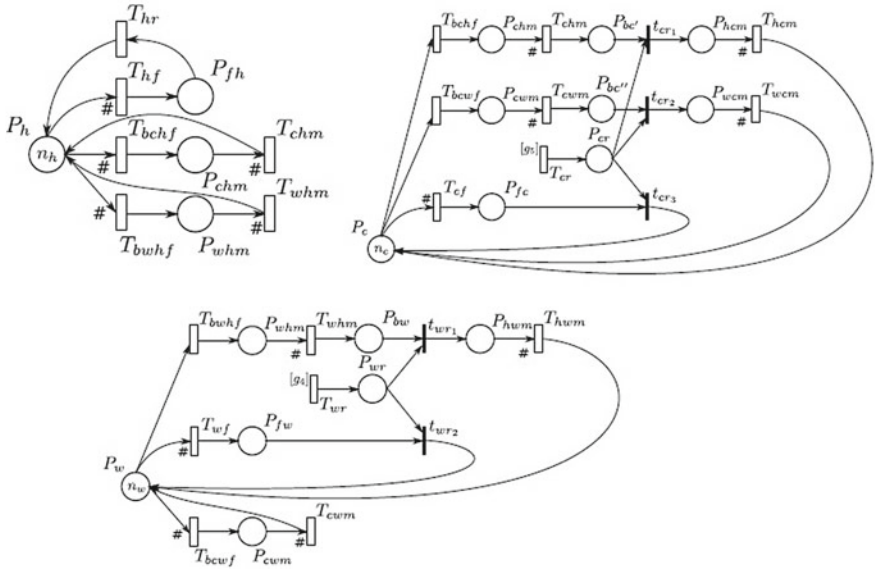

Fig. 22.17 Decomposed SRN availability model of IaaS cloud

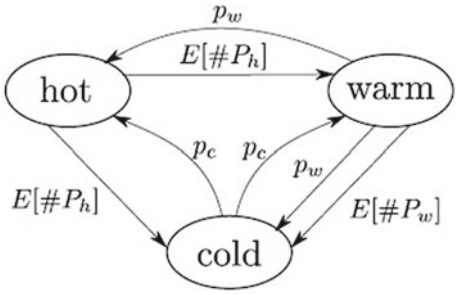

Fig. 22.18 Import graph describing sub-model interactions

solution to the overall problem can be set up as a fixed-point problem. Such problems can be solved iteratively by successive substitution with some initial starting point. Many mathematical issues arise such as the existence of the fixed point, the uniqueness of the fixed-point, the rate of convergence, accuracy, and scalability. Except for the existence of the fixed point [30], all other issues are open for investigation. Nevertheless, the method has been successfully utilized on many real problems [1].

Table 22.4 shows the effect of the decomposition/fixed-point iteration method (which is also known as interacting sub-models method), comparing the number of states of the monolithic model (column 2) with the number of states of the interacting sub-model case (column 3).

Many more examples of this type of multi-level models can be found in the literature [1, 2, 16, 29–35]. A particular example is the implementation of the Session Initiation Protocol (SIP) by IBM on its WebSphere. A hierarchical availability model of that system is described in detail in [16].

22.5 Relaxing the Exponential Assumption

One standard complaint about the use of homogeneous continuous-time Markov chains is the ubiquitous assumption of all event times being exponentially distributed. There are several known paradigms that can remove this assumption: non-homogeneous Markov chain, semi-Markov and Markov regenerative process, and the use of phase-type expansions. All these techniques have been used, and many examples are illustrated in [1].

Nevertheless, there is additional complexity in using non-exponential techniques in practice, partly because the analytical–numeric solution is more complex but also because of additional information about the non-exponential distributions which is then needed and is often hard to come by.

A flowchart comparing the modeling power of the different state-space model types is shown in Fig. 22.19 [1], and in Fig. 22.20, we provide a classification of the modeling formalisms considered in [1].

22.6 Conclusions

We have tried to provide an overview of known modeling techniques for the reliability and availability of complex systems. We believe that techniques and tools do exist to capture the behavior of current-day systems of moderate complexity. Nevertheless, higher and higher complexity is being designed into systems, and hence the techniques must continue to evolve. Together with the largeness problem, the need for higher fidelity will require increasing use of non-exponential distributions, the need to properly combine performance, power, and other measures of system effectiveness together with failure and recovery. Parameterization and validation of the

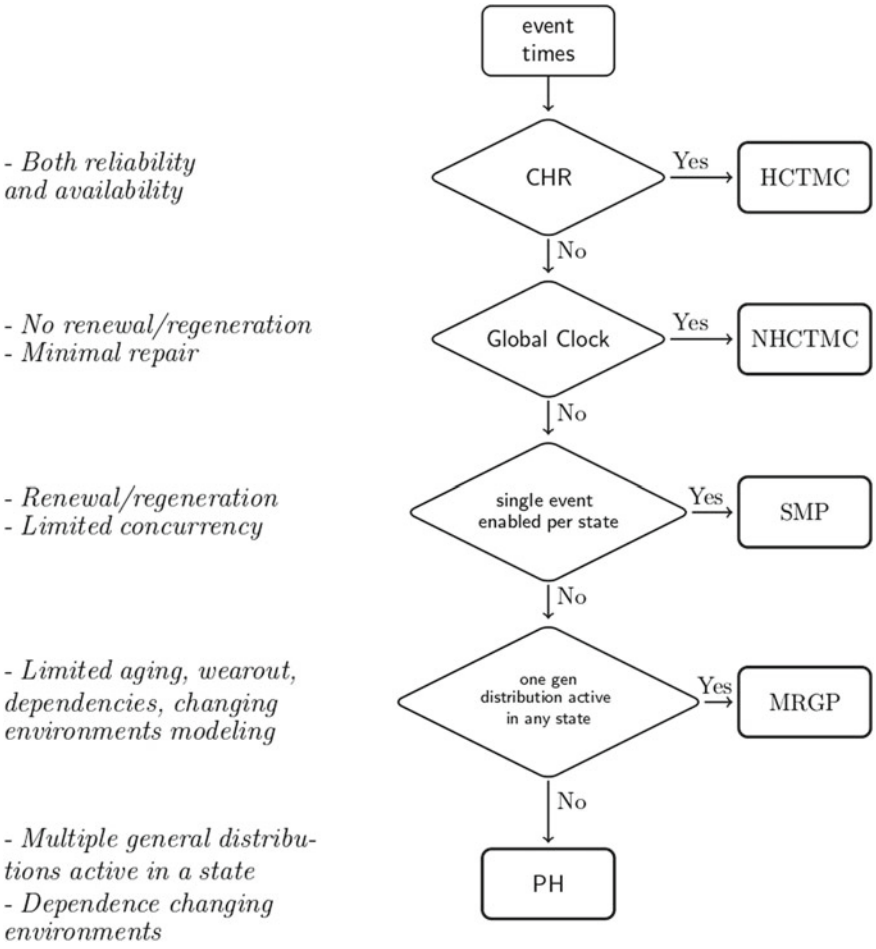

Fig. 22.19 Flow chart comparing the modeling power of the different state space model types [1]

models need to be further emphasized and aided. Tighter connection between data-driven and model-driven methods on the one hand, and combining simulative solution with analytic–numeric solution on the other hand, is desired. Validated models need to be maintained throughout the life of a system so that they can be used for tuning at operational time as well. Besides system-oriented measures such as reliability and availability, user-perceived measures need to be explored [34–36]. Uncertainty in model parameters, so-called epistemic uncertainty, as opposed to aleatory uncertainty already incorporated in the models discussed here, needs to be accounted for in a high-fidelity assessment of reliability and availability [37]. For further discussion on these topics, see [1].

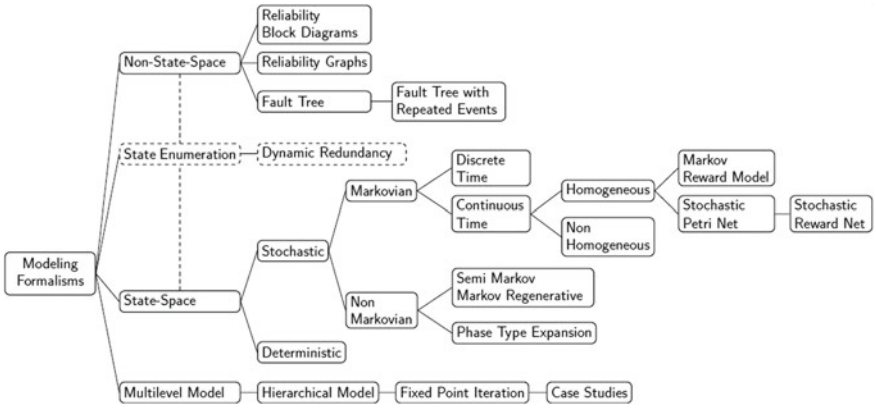

Fig. 22.20 Modeling formalisms

References

1. Trivedi, K., & Bobbio, A. (2017). *Reliability and availability engineering*. Cambridge: Cambridge University Press.
2. Sahner, R., Trivedi, K., & Puliafito, A. (1996). *Performance and reliability analysis of computer systems: An example-based approach using the SHARPE software package*. Kluwer Academic Publishers.
3. Trivedi, K., & Sahner, R. A. (2009). SHARPE at the age of twenty two. *ACM Performance Evaluation Review, 36*(4).
4. Ciardo, G., Muppala, J., & Trivedi, K. (1989). SPNP: Stochastic petri net package. In *Proceedings of Third International Workshop on Petri Nets and Performance Models* (pp. 142–151).
5. Hirel, C., Tuffin, B., & Trivedi, K. (2000). SPNP: Stochastic petri nets. Version 6. In B. Haverkort & H. Bohnenkamp (Eds.), *International Conference on Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS 2000), LNCS 1786* (pp. 354–357). Berlin: Springer.
6. Trivedi, K., Vasireddy, R., Trindade, D., Nathan, S., & Castro, R. (2006). Modeling high availability systems. In *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*.
7. Sebastio, S., Trivedi, K., Wang, D., & Yin, X. (2014). Fast computation of bounds for two-terminal network reliability. *European Journal of Operational Research, 238*(3), 810–823.
8. Ramesh, V., Twigg, D., Sandadi, U., Sharma, T., Trivedi, K., & Somani, A. (1999). An integrated reliability modeling environment. *Reliability Engineering and System Safety, 65,* 65–75.
9. Zang, X., Sun, H., & Trivedi, K. (2000). *A BDD-based algorithm for reliability graph analysis.* Department of Electrical & Computer Engineering: Duke University, Technical Report.
10. Soh, S., & Rai, S. (2005). An efficient cutset approach for evaluating communication-network reliability with heterogeneous link-capacities. *IEEE Transactions on Reliability, 54*(1), 133–144.
11. Malhotra, M., & Trivedi, K. (1994). Power-hierarchy among dependability model types. *IEEE Transactions on Reliability, R-43*, 493–502.
12. Zang, X., Wang, D., Sun, H., & Trivedi, K. (2003). A BDD-based algorithm for analysis of multistate systems with multistate components. *IEEE Transactions on Computers, 52*(12), 1608–1618.
13. Zang, X., Sun, H., & Trivedi, K. (1999). A BDD-based algorithm for reliability analysis of phased mission systems. *IEEE Transactions On Reliability, 48*(1), 50–60.

14. Merle, G., Roussel, J., Lesage, J., & Bobbio, A. (2010). Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events. *IEEE Transactions on Reliability, 59*(1), 250–261.
15. Trivedi, K. (2001). *Probability & statistics with reliability, queueing & computer science applications* (2nd ed.). Wiley.
16. Trivedi, K., Wang, D., Hunt, J., Rindos, A., Smith, W. E., & Vashaw, B. (2008). Availability modeling of SIP protocol on IBM © Websphere ©. In *Proceedings of Pacific Rim International Symposium on Dependable Computing (PRDC)* (pp. 323–330).
17. Fricks, R., Bobbio, A., & Trivedi, K. (2016). Reliability models of chronic kidney disease. In *Proceedings IEEE Annual Reliability and Maintainability Symposium* (pp. 1–6).
18. United States Renal Data System. (2014)."2014 annual data report: An overview of the epidemiology of kidney disease in the United States. National Institutes of Health—National Institute of Diabetes and Digestive and Kidney Diseases, Tech. Rep., 2014.
19. Stewart, W. (1994). *Introduction to the numerical solution of markov chains.* Princeton University Press.
20. Reibman, A., & Trivedi, K. (1988). Numerical transient analysis of Markov models. *Computers and Operations Research, 15*:19–36.
21. Reibman, A., Smith, R., & Trivedi, K. (1989). Markov and Markov reward model transient analysis: An overview of numerical approaches. *European Journal of Operational Research, 40*, 257–267.
22. Bobbio, A., & Premoli, A. (1982). Fast algorithm for unavailability and sensitivity analysis of series-parallel systems. *IEEE Transaction on Reliability, R-31*, 359–361.
23. Blake, J., Reibman, A., & Trivedi, K. (1988). Sensitivity analysis of reliability and performability measures for multiprocessor systems. *ACM SIGMETRICS Performance Evaluation Review, 16*(1), 177–186.
24. Matos, R., Maciel, P., Machida, F., Kim, D. S., & Trivedi, K. (2012). Sensitivity analysis of server virtualized system availability. *IEEE Transactions on Reliability, 61*, 994–1006.
25. Bobbio, A. (1990). System modelling with petri nets. In A. Colombo & A. de Bustamante (Eds.), *System reliability assessment* (pp. 103–143). Kluwer Academic P.G.
26. Ciardo, G., Muppala, J., & Trivedi, K. (1991). On the solution of GSPN reward models. *Performance Evaluation, 12,* 237–253.
27. Ciardo, G., Blakemore, A., Chimento, P., Muppala, J., & Trivedi, K. (1993). Automated generation and analysis of Markov reward models using stochastic reward nets. In C. Meyer & R. Plemmons (Eds.), *Linear algebra, markov chains, and queueing models, The IMA Vol in mathematics and its applications* (Vol. 48, pp. 145–191). Berlin: Springer.
28. Ghosh, R., Longo, F., Frattini, L., Russo, S., & Trivedi, K. (2014). Scalable analytics for IaaS cloud availability. *IEEE Transactions on Cloud Computing.*
29. Ciardo, G., & Trivedi, K. (1993). A decomposition approach for stochastic reward net models. *Performance Evaluation, 18,* 37–59.
30. Mainkar, V., & Trivedi, K. (1996). Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative models. *IEEE Transactions on Software Engineering, 22*(9), 640–653.
31. Sukhwani, H., Bobbio, A., & Trivedi, K. (2015). Largeness avoidance in availability modeling using hierarchical and fixed-point iterative techniques. *International Journal of Performability Engineering, 11*(4), 305–319.
32. Ghosh, R., Longo, F., Naik, V., & Trivedi, K. (2013). Modeling and performance analysis of large scale IaaS clouds. *Future Generation Computer Systems, 29*(5), 1216–1234.
33. Ghosh, R., Longo, F., Xia, R., Naik, V., & Trivedi, K. (2014). Stochastic model driven capacity planning for an infrastructure-as-a-service cloud. *IEEE Transactions Services Computing, 7*(4), 667–680.
34. Trivedi, K., Wang, D., & Hunt, J. (2010). Computing the number of calls dropped due to failures. *ISSRE*, 11–20.
35. Mondal, S., Yin, X., Muppala, J., Alonso Lopez, J., & Trivedi, K. (2015). Defects per million computation in service-oriented environments. *IEEE Transactions Services Computing, 8*(1), 32–46.

36. Wang, D., & Trivedi, K. (2009). Modeling user-perceived service reliability based on user-behavior graphs. *International Journal of Reliability, Quality and Safety Engineering, 16*(4), 1–27.
37. Mishra, K., & Trivedi, K. (2013). Closed-form approach for epistemic uncertainty propagation in analytic models. In *Stochastic reliability and maintenance modeling* (Vol. 9, pp. 315–332). Springer Series in Reliability Engineering.

Dr. Kishor Trivedi is a Professor of Electrical and Computer Engineering and Computer Science at Duke University and a Life Fellow of IEEE. He is the author of a well-known text entitled, *Probability and Statistics with Reliability, Queuing and Computer Science Applications.* His latest book, co-authored with Andrea Bobbio, *Reliability and Availability Engineering*, is published by Cambridge University Press in 2017. He has supervised 48 Ph.D. dissertations and has an h-index 100 as per Google scholar. As a recipient of IEEE Computer Society's Technical Achievement Award for research on Software Aging and Rejuvenation, he has worked closely with industry in carrying out reliability/availability analysis and in the development and dissemination of modeling software packages such as HARP (with NASA), SAVE (with IBM), SHARPE, SPNP, and Boeing's IRAP.

Dr. Andrea Bobbio is a Professor of Computer Science at Università del Piemonte Orientale in Italy and Senior Member of IEEE. His academic and professional activity has been mainly in the area of reliability engineering and system reliability. He contributed to the study of heterogeneous modeling techniques for dependable systems, ranging from non-state-space techniques to Bayesian belief networks, to state-space-based techniques, and fluid models. He has visited several important institutions and is the author of 200 papers in international journals and conferences. He is co-author of the book, *Reliability and Availability Engineering*, published by Cambridge University Press in 2017.