Models, Methods and Tools for Availability Assessment of IT-Services and Business Processes

Dr.-Ing. Nikola Milanovic

Habilitationsschrift an der Fakultät IV –
Elektrotechnik und Informatik
der Technischen Universität Berlin

Lehrgebiet:
Informatik

Eröffnug des Verfahrens: 15. Juli 2009
Verleihung der Lehrbefähigung: 19. Mai 2010
Ausstellung der Urkunde: 19. Mai 2010

Gutachter:
Prof. Dr. Volker Markl (Technische Universität Berlin)
Prof. Dr. Miroslaw Malek (Humboldt Universität zu Berlin)
Prof. Dr. Alexander Reinefeld (Humboldt Universität zu Berlin)

Berlin, 2010

D 83

Abstract

In the world where on-demand and trustworthy service delivery is one of the main preconditions for successful business, service and business process availability is of paramount importance and cannot be compromised. For that reason service availability is in central focus of the IT operations management research and practice. This work presents foundations, models, methods and tools that can be used for comprehensive service and business process availability assessment and management.

As many terms in this emerging field are used colloquially, Chapter 2 provides detailed background and definitions. The problem of service availability assessment and management is interdisciplinary, combining fields of fault-tolerance, service oriented architecture and business process management. Another role of this chapter is to make the text accessible to readers with different backgrounds. After the context of service availability has been introduced, Chapter 3 presents existing models for availability and performability assessment. The emphasis is on formal availability models, such as combinatorial (e.g., reliability block diagrams, fault trees) and state-space (e.g., Markov chains, Petri net) models, but qualitative models (e.g., maturity models such as ITIL or CobiT or standards such as ISO 27002 and ISO 12207) are also covered, albeit with limited scope as they are not the primary focus of this work. In Chapter 4, more than 60 commercial, public domain and academic tools for availability assessment are surveyed and compared. Downsides and limitations of standard availability models, both from methodical and practical perspective, are identified and a novel approach for quantitative service availability assessment is presented in Chapter 5. It treats service and process availability as complex functions of availability properties of underlying ICT infrastructure elements and enables automatic generation of availability models, based on service or process description. Finally, Chapter 6 positions presented work in the context of a comprehensive vision for model-based IT service management.

Acknowledgements

First and foremost I would like to thank my supervisors, Prof. Miroslaw Malek (Humboldt University Berlin) and Prof. Volker Markl (Berlin University of Technology), for their continued support throughout my work. I am thankful to Prof. Andras Pataricza (University of Budapest) and his group for fruitful cooperation. My thanks also go to numerous friends and colleagues, among others Reinhard Meyer and Stefan Brüning, who helped with the survey of tools for availability modeling, and specially to Bratislav Milic who contributed significantly to research in the area of service mapping and infrastructure description.

I would like to express my gratitude to Dr. Steve Flanagan (IsoGraph) and Dr. Uwe Laude (Federal Office for Information Security / Bundesamt für Sicherheit in der Informationstechnik) for supporting this research and making IsoGraph and GSTool software packages available to me. Prof. Kishor Trivedi (Duke University) and Prof. William Sanders (University of Illinois) deserve additional recognition for providing me with SHARPE and Möbius software.

This work would not be possible without the patience, support and love of my son Nebojsa and my wife Maja.

Berlin, 03.06.2009

Contents

List of Figures

Chapter 1

Introduction

Service oriented architecture (SOA) is the paradigm that aspires to play a dominant role in shaping of the Information Technology (IT) landscape. It represents a logical and evolutionary step initiated by developments in areas of distributed computing, business process modeling, and increased ubiquity of networking technologies. The main goal of SOA is to introduce standard methodologies, architectures, tools, languages and protocols for development and integration of distributed applications based on loosely coupled, independent and autonomous software artifacts, thus supporting the large-scale composability, reusability and agility. As discussed in Chapter 2 in more detail, SOA is usually realized today by frameworks and technologies such as SOAP[1]- or REST[2]-based Web Services.

Dynamic and trustworthy service delivery in SOA is at present one of the main preconditions for successful and sustainable business operations. Service and business process availability is, therefore, of paramount importance and cannot be compromised. Even today, services are simply expected to be delivered reliably and on demand, and this requirement will be even more important in the near future. Unreliable and incorrect services can corrupt business processes causing an impact ranging from lost opportunity or money to loss of lives. Common understanding of service and business process availability properties is rather sketchy, limited and mostly empirical. Regardless of this fact, services are already widely used as the new paradigm to build software architectures and enable complex business processes. According to the IEEE Technical Committee on Services Computing, services now account for more than 75% of developed economies.

Several methodologies can be used to assess service and business process availability: quantitative, qualitative and analytical. Quantitative assessment is based on real-time measurement and monitoring. Whereas it has proven itself in several areas (e.g., hardware benchmarks and testing), it is difficult to apply to services because of the lack of adequate metrics and instrumentation. Qualitative availability assessment is performed informally (e.g., through questionnaires and interviews) and assigns an availability class to the system (service). Qualitative results are easy to misinterpret, difficult to compare and

[1] formerly known as Simple Object Access Protocol
[2] Representational State Transfer

depend heavily on a consultant who is performing the analysis. Analytical methods are used to model services and their behavior and calculate or simulate their availability. Up to now, however, classical analytical methods have been applied to determine service availability with mixed success and relative low industry penetration due to scalability, complexity and evolution problems.

With the recent advance of cloud computing (e.g., Amazon EC-2) and software as a service (e.g., Google Applications) paradigms, importance of the ability to model and assess service availability has only increased. For example, in recent years Google Mail and Google Applications services have experienced increasing number of outages [35], some of them lasting well over 24 hours [162]. Consequently, fears were raised that inability to provide strictly defined service level agreement in terms of the maximum number of downtime hours per year may lead to rejection of the software as a service (SaaS) paradigm [163]. Many Chief Technology Officers (CTO) even speculated that they consider returning to locally hosted solutions, where SaaS products such as Google Mail may be used as a backup application only. Many CTOs have used the availability vocabulary in their statements (e.g., specifying the number of maximum downtime hours they were ready to tolerate) without being explicitly aware of it. The Amazon EC-2 cloud computing service also experiences periodic long-term outages. Let us examine one official EC-2 post-mortem report, dated April 8, 2007 [197]:

> The network event had two phases described below.
>
> During the first phase, which lasted from approximately 23:45 PDT yesterday until 01:15 PDT today, network connectivity was significantly degraded for many of our instances. These initial connectivity issues were caused by a degradation of part of our routing infrastructure. Normally, this sort of event would have failed over quickly and automatically, but this particular failure mode did not trigger the failover mechanism and instead remained in degraded operation. We are working on fully understanding this failure mode and updating our network to handle it correctly.
>
> Failover eventually happened at 01:15 PDT, and external network connectivity was fully restored to most instances. A suboptimal configuration caused a much smaller number of instances to regain connectivity more slowly after the failover. We have a fix to this configuration to avoid this delayed recovery in the future.

This report vividly illustrates inability to model availability of complex service oriented systems satisfactorily, reflected in the claim that failover should have happened, but did not due to unrecognized failure mode. It further indicates that availability of a large-size SaaS system is a complex function of many parameters, including the underlying network infrastructure (routers), configuration and the people maintaining it. Finally, it confirms that pure reliability/availability models do not describe complex SOA systems completely, because services may be still functioning after the fault, albeit with reduced

quality of service (performance). Therefore, issues of service performability and degraded performance have also to be addressed.

Service availability is but one of many attributes which can constitute different Quality of Service (QoS) measures. Figure 1.1 shows the "big picture", where services express QoS measures such as performability, dependability or security. Each QoS measure is composed of attributes. For example, dependability is composed of availability, reliability, safety, integrity and maintainability, while security is composed of availability, confidentiality and integrity (these compositions are not given in Figure 1.1 explicitly). We will not be investigating any of these QoS measures completely. Instead, focus of this work is the service reliability/availability attribute and, to some extent, how it influences service performability. However, we will also not discuss service performance, which together with reliability/availability constitutes performability. This will necessary limit our discussions on service performability, as we will assume that service performance model already exists (e.g., stochastic queuing models).

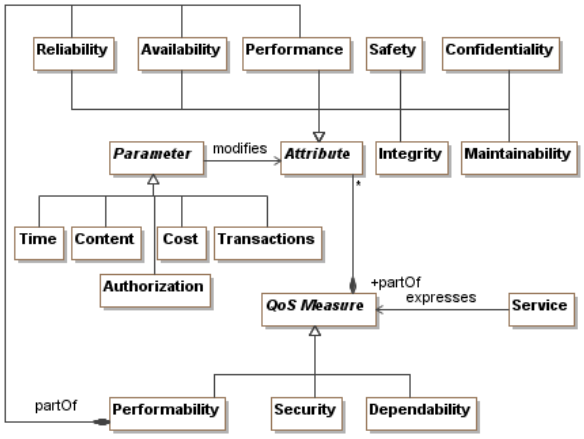

Figure 1.1: Service QoS measures and attributes

The goal of this work is therefore to investigate applicability of the existing methodologies and tools to the problem of service availability assessment, and to propose a novel approach eliminating some of the issues that will be identified. The manuscript is structured as follows. In Chapter 2 basic definitions concerning availability, services and business processes are given. It is then conjectured that there exists neither satisfactory definition of service availability nor satisfactory model that enables its effective assessment and management. Chapter 3 introduces standard models and methods for availability assessment, followed by Chapter 4 which surveys existing tools supporting (service) availability modeling. In Chapter 5 inadequacies of surveyed models, methods and tools for service availability assessment are identified and a novel method is proposed, which is based on the mapping of service and process activities to underlying ICT-components, enabling automatic availability model generation. Finally, Chapter 6 presents a summary and places the work in the context of a comprehensive model-based IT service management vision.

Chapter 2

Definitions

In this chapter our discourse will be described, starting with basic definitions of reliability and availability, known from the theory of fault-tolerant (computer) systems. We will then define services and business processes, and finally construct a framework (fault model, to be more precise), in which we will apply availability definitions to service-oriented systems. Definitions given here will be used further throughout this work.

2.1 Reliability and Availability

Numerous definitions of reliability and availability exist. We will summarize the most important ones and then distill them in a definition that is compatible with the purpose of this manuscript. In order to be able to define reliability and availability, however, we first need to introduce basic probability theory definitions.

The events that lead to the system malfunction have intrinsic probabilistic nature. Therefore, the lifetime or time to failure of a system can usually be represented by a random variable. A phenomenon is considered random if its future behavior is not exactly predictable. An example is tossing a pair of dice or measuring the time between particle emissions by a radioactive sample. A function that associates a number with every possible outcome of an event is called a random variable.

Let the random variable X represent the lifetime or time to failure of a system. The continuous random variable X can be characterized by the cumulative distribution function $F(t)$, the probability density function $f(t)$ and the hazard rate function $h(t)$, also known as the instantaneous failure rate. The cumulative distribution function (CDF) represents probability that the system will fail before a given time t:

$$F(t) = Pr(X \leq t)$$

The probability density function (PDF) describes the rate of change of the CDF and for continuous random variables is given as:

$$f(t) = \frac{dF(t)}{dt} = \lim_{\Delta t \to 0} \frac{Pr(t < X \leq t + \Delta t)}{\Delta t}$$

It can be seen that $f(t)\Delta t$ is the limiting probability that a system will fail in the interval $(t, t+\Delta t)$. This probability is unconditional. If we, however, observe a system that was functioning up to some time t, the conditional probability for this interval will be different from $f(t)\Delta t$. Therefore we define instantaneous failure rate, or the hazard rate function:

$$h(t) = \lim_{\Delta t \to 0} \frac{Pr(t < X \leq t + \Delta t | X > t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{Pr(t < X \leq t + \Delta t)}{\Pr(X > t)\Delta t} = \frac{f(t)}{1 - F(t)}$$

The $h(t)\Delta t$ represents the conditional probability that a system that has survived until time t will fail in the interval $(t, t + \Delta t]$. The following equations describe relationships between the CDF, PDF and the hazard rate function:

$$f(t) = \frac{dF(t)}{dt} = h(t)e^{-\int_0^t h(\tau)d\tau}$$

$$F(t) = \int_0^t f(\tau)d\tau = 1 - e^{-\int_0^t h(\tau)d\tau}$$

$$h(t) = \frac{f(t)}{\int_t^\infty f(\tau)d\tau} = \frac{dF(t)/dt}{1 - F(t)}$$

Two very important properties used to describe random variable X are the mean or expected value $E[X]$ and the variance σ_X^2. For the continuous random variable X the mean is given as:

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

The variance is defined as:

$$\sigma_X^2 = E[(x - E[x])^2]$$

The square root of the variance, σ_X is called the standard deviation.

Time to failure or lifetime of a system may follow different distributions, some of which will be investigated. The exponential distribution is one of the most commonly encountered and used in reliability models of hardware and software. PDF, CDF and hazard functions for the exponential distribution are:

$$f(t) = \lambda e^{-\lambda t}$$
$$F(t) = 1 - e^{-\lambda t}$$
$$h(t) = \lambda$$

The parameter λ is called the failure rate as it describes the rate at which failures occur in time. It is usually assumed that λ is constant, but in reality

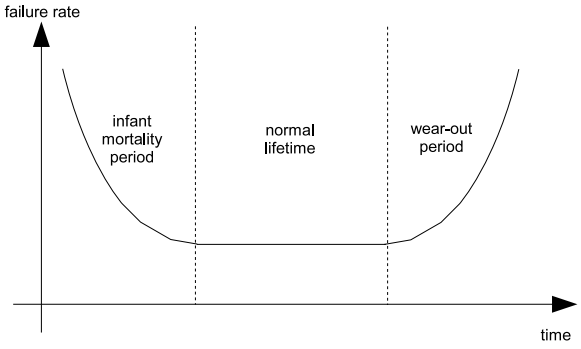

Figure 2.1: Failure rate as a function of time (the bathtub curve)

it follows the bathtub-shaped curve (Fig. 2.1). The mean and the standard deviation of the exponential distribution are $1/\lambda$.

In the early life of a system, failure rate is high (infant mortality period) and then decreases while the system is entering the useful life part. The failure rate is constant in this period. Wearout period describes the end of the specified system's lifetime, where failure rate begins to grow again. The bathtub curve is usually applied directly to hardware, but can be assumed that it also holds for the software components. The more difficult issue is the problem of software upgrades and patches, but also of rejuvenation and aging (see the Note on Software Reliability at the end of Section 2.1.1). Frequently, other distributions are used to model failure behavior of the software systems, such as normal, gamma or Bayesian distributions.

Weibull distribution, often used due to its configurability, has two parameters: α (the shape parameter) and λ (failure rate or the scale parameter). PDF, CDF and hazard functions for the Weibull distribution are given as:

$$f(t) = \alpha\lambda(\lambda t)^{\alpha-1}e^{-(\lambda t)^{\alpha}}$$

$$F(t) = 1 - e^{-(\lambda t)^{\alpha}}$$

$$h(t) = \alpha\lambda(\lambda t)^{\alpha-1}$$

The mean of the Weibull distribution is $\mu = \Gamma((\alpha+1/\alpha)/\lambda$ and the standard deviation is $\sigma = [\Gamma((\alpha+2)/\alpha) - \Gamma^2((\alpha+1)/\alpha)]^{1/2}/\alpha$, where the gamma function $\Gamma(\omega)$ is given as:

$$\int_0^{\infty} \rho^{\omega-1} exp(-\rho) d\rho$$

If the time t takes only discrete units 0,1,2..., then the geometric distribution corresponds to the continuous time exponential distribution. It is easily obtained from exponential distribution, replacing $exp(-\lambda)$ by q, which is the failure probability rate ($0 < q < 1$), and t by n. The probability mass function and CDF of the geometric distribution are given as:

$$f(n) = q^n(1-q)$$
$$F(n) = 1 - q^n$$

The mean and the standard deviation of this distribution are $\mu = 1/(1-q)$ and $\sigma = q^{1/2}/(1-q)$

Analogously to the geometric distribution, the discrete Weibull distribution's probability mass function, CDF, and hazard function are given as:

$$f(n) = q^{n^\alpha}(1 - q^{(n+1)^\alpha - n^\alpha})$$
$$F(n) = 1 - q^{n^\alpha}$$
$$h(n) = 1 - q^{(n+1)^\alpha - n^\alpha}$$

The mean of this distribution ($\sum_{k=0}^{\infty} q^{k^\alpha}$) is very difficult to derive in a closed form for any given q and α.

Using these preliminaries, we will now introduce reliability and availability definitions.

2.1.1 Reliability

One of standard reliability definitions is the recommendation E.800 of the International Telecommunications Union (ITU-T) which defines reliability as

> the ability of an item to perform a required function under given conditions for a given time interval. [207]

In [186] reliability is defined as

> a function of time, $R(t)$, representing the conditional probability that the system has survived the interval $[0, t]$, given that it was operational at time $t = 0$.

In the seminal work in the field of fault-tolerance and dependability [11], reliability is defined as the attribute of dependability. Dependability is the ability to deliver the service that can justifiably be trusted. It has the following attributes: availability, reliability, safety, integrity and maintainability. Hence, in this context reliability is perceived as

> the continuity of correct service.

Finally, in [93], reliability is defined as

> the function $R(t)$, which is the probability that the system continues to function until time t.

and similarly in [173] as

> the probability that the system is able to function correctly over a specified period of time.

We will adopt the following definition of reliability:

Def. 1 *For any time interval* $(z, z+t]$*, reliability function* $R(t|z)$ *is the conditional probability that the system does not fail in this interval, assuming it was working at time* z*. We will observe intervals starting at* $z = 0$*, where reliability* $R(t) = R(t|0)$ *is the probability that the system continues to function until time* t*.*

Let the random variable X be the time to system failure. Reliability can then be expressed as:

$$R(t) = Pr(X > t) = 1 - F(t)$$

where F(t) is the CDF of the system lifetime X. We will not be considering systems that are defective to begin with, for which we would require mixture distributions for $F(t)$. We will also assume that no system can work indefinitely without faults:

$$\lim_{t\to\infty} R(t) = 0$$

Function $R(t)$ is a nonincreasing, continuous, and monotone with values ranging between 0 and 1 in the interval $[0, \infty)$. For the graphical representation of reliability, we integrate system lifetime PDF over time:

$$R(t) = \int_t^\infty f(x)dx$$

Reliability is the area under the curve $f(t)$ from t to infinity, as shown in Figure 2.2.

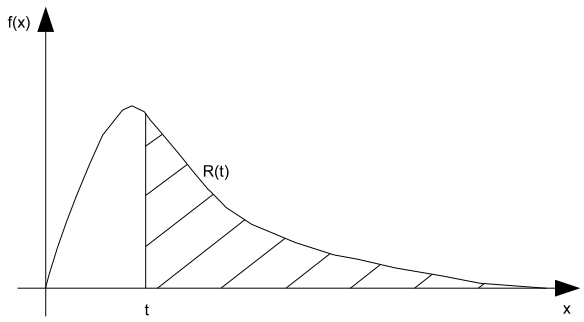

Figure 2.2: Graphical interpretation of reliability

Using the reliability definition, we can express the mean time to failure (MTTF), which represents the expected time that a system will operate before the first failure occurs. For example, the system will on the average operate for MTTF hours and then encounter its first failure. MTTF can be calculated as the mean of the system's lifetime distribution:

$$MTTF = E[X] = \int_0^\infty tf(t)dt = \int_0^\infty R(t)dt$$

For example, for exponentially distributed system's lifetime, reliability and MTTF are given as:

$$R(t) = 1 - F(t) = e^{-\lambda t}$$

$$MTTF = \int_0^\infty e^{-\lambda t} dt = \frac{1}{\lambda}$$

Note on Software Reliability. The bathtub curve from Figure 2.1 is empirically derived and mainly used for hardware (mechanical and electronic components). Research into software reliability has, up to date, not been performed up to comparable depth. However, several models attempting to express peculiarities of software reliability exist. Software reliability is usually defined as the probability of failure-free software operation for a specified period of time in a specified environment [132]. Failure rate is then defined for a given deployment environment and with respect to the cumulative run-time. Contrary to the bathtub curve, the software failure rate has testing/debug phase, useful life and obsolence, as illustrated in Figure 2.3. Industrial practice shows that failure rate is influenced by software updates (version upgrades, patches), which inevitably introduce new incorrect behavior, explaining the failure rate spikes.

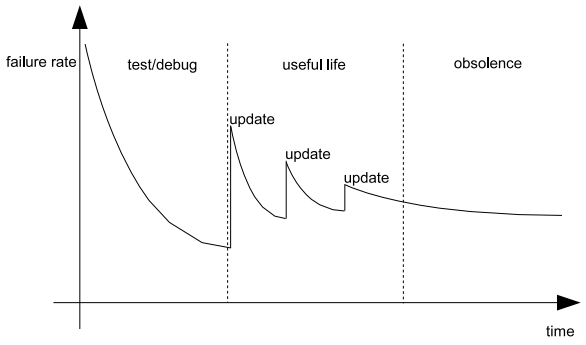

Figure 2.3: Software failure rate

Several models exist which describe software reliability and one of the first proposed was Jelinski-Moranda model [111]. It assumes that a hazard rate for failures is a piecewise constant function and that failure rate is proportional to the remaining number of errors:

$$h(t) = C(N - (i-1))$$

where C is the proportionality constant and N is the number of errors initially in the program. $h(t)$ is applied in the interval between detection of error $(i-1)$ and i. Hence the reliability function and mean time to failure are defined as:

$$R(t) = e^{-C(N-(i-1))t}$$

$$MTTF = \frac{1}{C(N-(i-1))}$$

Shooman's model [185] is similar to Jelinski-Moranda, as it assumes that hazard function is proportional to the number of remaining errors in the software. It further assumes that errors are removed as soon as they are discovered. The reliability function is given by:

$$R(t) = e^{-C\left(\frac{E_T}{I_T} - \frac{E_C}{I_T}\right)t}$$

where E_T is the number of errors initially present in the program, I_T the number of statements in the program and E_C the number of errors corrected so far. Mean time to failure is then given as:

$$MTTF = \frac{I_T}{C(E_T - E_C)}$$

Musa's model [152] introduced the notion that execution (processing) time should be the base for software reliability theory instead of calendar time. For him too, hazard function (failure rate) is proportional to the number of remaining errors. Execution time rate of change of the number of faults corrected is proportional to the hazard rate. Reliability in the Musa's model is:

$$R(t) = e^{(-fK(N-m))t}$$

where t is the execution time, f is the average instruction execution time divided by the total number of instructions, K is the proportionality constant, N the number of errors initially present and m the number of errors corrected so far.

Littlewood's model [130] is the example of Bayesian models which assume exponential distribution of failures with respect to the program's operating time. The error distribution is considered independent and the failure rate parameter is modeled as a random process. Errors are removed upon occurrence. The hazard rate function is given as:

$$h(t) = \frac{(N-i)a}{b + t + t_1}$$

where a and b are parameters of the Gamma distribution, t is time elapsed so far and t_1 execution time. The model proposes the following reliability function:

$$R(t) = (1 - \frac{b+t}{b+t_0+t_1}b + \frac{b+t_0}{b+t_0+t_1+t}a)(N-i)$$

Jelinski-Moranda, Shooman's and Musa's model are the examples of path-based models, where system reliability is computed by taking into account possible execution paths of the program, either experimentally by testing or algorithmically. Other examples of path-models are Krishnamurthy and Mahur model [119] or Yacoub model [220]. On the other side, Littlewood's model represents state-based models, which assume that the transfer of control between

components has a Markov property. The advantage of state-based models is that they can be used even when no source code is available. Other examples of state-based models are Cheung model [58], Laprie model [124], Kubat model [120], Gokhale model [87] or Ledoux model [125].

Software reliability measures can at present be achieved with pretty good accuracy if programming team has a substantial track data and lots of reliability data to support it, which is rarely the case. As no standard or widely accepted reliability model exists, curve fitting seems to be most popular in practice.

2.1.2 Availability

Availability is closely related to, but also very often confused with reliability. According to the ITU-T recommendation E.800 [207], availability is defined as

> the ability of an item to be in a state to perform a required function at a given instant of time, or at any instant of time within a given time interval, assuming that the external resources, if required, are provided.

In [186], availability is defined as:

> a function of time $A(t)$, which represents the probability that the system is operational at the instant of time t. If the limit of this function exists as t goes to infinity, it expresses the expected fraction of time that the system is available to perform useful computations.

Analogously to reliability, [11] treats availability as an attribute of dependability and defines it as:

> the readiness for the correct service.

Finally, in [173] availability is defined as:

> the probability that system is working at the instant t, regardless of the number of times it may have failed and been repaired in the interval $(0, t)$.

The main difference between reliability and availability is that reliability, as can be seen from its definition, refers to and requires failure-free operation of the system during any interval. It defines probability that no failures have occurred during the entire observed interval. Availability, on the other hand, focuses on the failure-free operation at a given instant of time, allowing that a system may have broken down in the past and has been repaired. If the system is not repairable however, definition and meaning of availability and reliability are equivalent. Let us introduce availability definition that we will be using and then formalize it.

Def. 2 *Instantaneous availability is the probability that the system is operational (delivers the satisfactory service) at a given time instant.*

Def. 3 *Steady state availability is a fraction of lifetime that the system is operational.*

Def. 4 *Interval availability is the probability that the system is operational (delivers satisfactory service) during a period of time.*

We can restate these definitions using random variables. Let $Y(t) = 1$ if the system is functioning at time t, and 0 otherwise. The instantaneous availability is the probability that $Y(t) = 1$, that is, it is equivalent to the mean of the random variable Y:

$$A(t) = Pr(Y(t) = 1) = E[Y(t)]$$

If we know $A(t)$ we can define the steady-state availability as:

$$A = \lim_{t \to \infty} A(t)$$

The steady-state availability is the long-term probability that the system is available. It can be shown that steady-state availability does not depend on the nature of the failure or repair time distribution, but only on the average time required to repair system failure and average time to system failure. This relationship is given as:

$$A = \frac{MTTF}{MTTF + MTTR}$$

The proof of this important formula can be found in [173], and we will also give another version in the Section 3.1.4 using Markov chains. The parameter MTTF is "mean time to failure" and represents the average time until the first failure occurs, and MTTR is "mean time to repair" and represents the average time required for repair, including any time to detect that there is the failure, to repair the failure, and place the system back into operational state. This means that, once the failure has occurred, it will take MTTR time units (e.g., hours) on the average to restore correct operations. There are two important assumptions with this formula. First, repairs can always be performed, and the system is then restored to its best condition. Second, the formula does not apply to systems with internal redundancy, but only to systems with a single up and a single down state.

If the system lifetime follows exponential distribution with the failure rate λ and the time to repair follows exponential distribution with the repair rate μ, then previous equation can be rewritten as:

$$A = \frac{\mu}{\lambda + \mu}$$

Typical availability function is given in Figure 2.4.

This shape applies to repairable systems. At the initial instant availability is 1, and then decreases converging towards a constant limit which is the steady-state availability. Of more interest is to know what happens within a given time interval. Thus we define interval (or average) availability as:

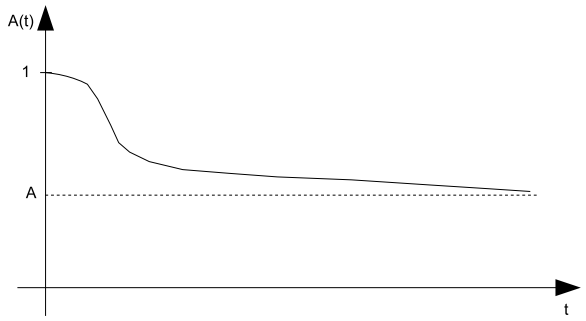

Figure 2.4: Typical availability function

$$A_I(t) = \frac{1}{t}\int_0^t A(\tau)d\tau$$

The $A_I(t)$ is the expected proportion of time the system is operational during the interval $(0, t]$. If total amount of the system uptime during this interval is the random variable $U(t)$, then:

$$A_I(t) = \frac{1}{t}\int_0^t E[Y(\tau)]d\tau = \frac{1}{t}E[U(t)]$$

The limiting interval (average) availability can also be defined:

$$A_I = \lim_{t\to\infty} A_I(t)$$

It can be shown that, if this limit exists, it is equal to steady state availability:

$$A_I = \lim_{t\to\infty} \frac{1}{t}\int_0^t A(\tau)d\tau = A$$

To achieve a better understanding of terms such as MTTF, MTTR, reliability and availability, let us observe a real-world example where availability of a multi-component Blade server system was analyzed [188]. Table 2.1 captures the upper and lower bounds for mean time to failure of field replaceable units, such as fans, power supplies, base blades, etc.

Based on these values (and on the complex availability model not shown here), different availability aspects can be calculated, such as component contribution to the single blade downtime (Table 2.2) or availability and downtime of a chassis containing up to 14 blades (Table 2.3).

The example also shows that availability is frequently expressed as the logarithmic unit called the "number of nines", which is then easily translated into downtime (another frequent and popular way to express availability) for the given time interval (usually a year). Similar studies can be found in many other areas, for example, in the field of hard disk drive reliability and availability we recommend some recent large-scale studies such as [164] or [181].

Field replaceable unit	Low MTTF	High MTTF
Fibre channel switch	320,000	440,000
Power supply	670,000	910,000
Fan	3,100,000	4,200,000
Midplane	310,000	420,000
Ethernet daughter card	6,200,000	8,400,000
Fibre Channel Daughter Card	1,300,000	1,800,000
Hard Disk Drive	200,000	350,000
2GB Memory Bank (2 DIMMs)	480,000	660,000
CPU	2,500,000	3,400,000
Base Blade	220,000	300,000
Ethernet Switch	120,000	160,000

Table 2.1: Typical reliability ranges (in hours) [188]

Component	High MTTF	Low MTTF
Software	14.99996	14.99957
Chassis Midplane	0.86087	0.63885
Blade CPU	0.41838	0.30763
Blade Memory	2.19000	1.59273
Chassis Power Subsystems	0.04839	0.03563
Chassis Cooling	0.00000	0.00000
Blade Disk Drive	7.88400	2.10240
Blade Base + Network Switches	7.16634	5.25582
Total	33.56794	24.93263

Table 2.2: Component contributions to blade downtime (hours/year) [188]

Number of blades	Number of spare blades	Availability	Downtime
14	0	0.999240794	399.04
13	1	0.999998166	0.96
12	2	0.999998433	0.82
11-7	3-7	0.999998433	0.82
6-1	8-13	0.999998511	0.78

Table 2.3: Availability and downtime (hours) of a 14-blade chassis [188]

2.2 Performability

The implicit assumption in reliability and availability analysis is that relevant system states are binary: either the system is functional or not. However, under partial failures the system's performance degrades, but the functionality remains. This class of systems is called degradable, that is, depending on the history of the system's structure, internal state and the environment during a specified utilization period T, the system will exhibit one of several worthwhile levels of performance. In this context, pure performance evaluation (of the fault-free system) will generally not suffice since structural changes

caused by faults may change (degrade) performance. Similarly, pure reliability/availability models no longer suffice since fault models where success can take only binary form (functioning or failed) cannot express systems that operate with degraded performance (the only correct/success state is the absence of a failure).

Therefore, performability is introduced as a metric which answers the question how "good" can a fault-tolerant system behave under the presence of faults, and how faults influence its functionality in terms of performance. In [143], performability is defined as

> a metric which relates directly to system effectiveness and is a proper generalization of both performance and reliability. A critical step in performability modeling is the introduction of the concept of a "capability function" which relates low-level system behavior to user-oriented levels of performance.

In [203] gracefully degradable systems (and performability) are defined as

> having redundant components that are all used at the same time to increase the system processing power.

In [131] performability is described as a metric which determines and quantifies degradable system's ability to

> detect the failure and reconfigures if a component fails, reaching a degraded state of operation in which it continues to provide service but at a reduced capacity. A degradable system can have several reduced operational states between being fully operational and having completely failed. Each state provides some performance level.

We will formalize the notion of both performance and performability. Let us refer to the fault-tolerant system and its environment as the total system. The probability space (Ω, ε, P) underlines the total system, where Ω is the sample space, ε is a set of events (measurable subsets of Ω) and $P : \varepsilon \rightarrow [0, 1]$ is the probability measure.

Let S denote one such total system, where S comprises a fault-tolerant system C and its environment E. The behavior of S can be described as a stochastic process $X_S = \{X_t | t \in T\}$, where T is a set of real numbers (observation times) called the utilization period, and for all $t \in T$, X_t is a random variable $X_t : \Omega \rightarrow Q$ defined on the underlying description space and taking values in the state space Q of the total system. The state space Q is the Cartesian product of the states sets of the fault-tolerant system and its environment: $Q = Q_C \times Q_E$. The stochastic process X_S is referred to as the base model of S. An instance of the base model's behavior for a fixed $\omega \in \Omega$ is a state trajectory $u_\omega : T \rightarrow Q$ where $u_\omega(t) = X_t(\omega) \forall t \in T$.

In formal terms, the user-oriented view of a total system's behavior is also defined in terms of the underlying probability space. We assume that the user is interested in distinguishing a number of different levels of accomplishment when judging how well the system has performed throughout the utilization

period. The user's description space is defined thus with an accomplishment set L whose elements are referred to as accomplishment levels or performance levels. In this context, the system performance can be formally defined as a random variable:

$$Y_S : \Omega \to L$$

where $Y_S(\omega)$ is the accomplishment level corresponding to outcome ω in the underlying description space.

Based on these preliminaries, let us define performability.

Def. 5 *Let B be a total system with performance Y_S taking values in accomplishment set L. The performability p_S of S is the function where for each measurable set B of accomplishment levels ($B \subseteq L$), the following holds:*

$$p_S(B) = P(\{\omega | Y_S(\omega) \in B\})$$

In other words, since P is the probability measure of the underlying probability space, for a designated set B of accomplishment levels, performability $p_S(B)$ is the probability that S performs at a level from B. If the performance Y_S is a continuous random variable, probability is uniquely determined by the probability distribution function of Y_S, defined for all $b \in L$:

$$F_S(b) = P(\{\omega | Y_S(\omega) \leq b\})$$

The performability can now be defined as:

$$p_S(B) = \int_B dF_S(b)$$

If Y_S is a discrete random variable, then p_S is uniquely determined by the probability distribution of Y_S, that is, by the set of values $\{p_S(a) | a \in L\}$.

Def. 6 *If S is a total system with performance Y_S taking values in accomplishment set L and Y_S is a discrete random variable, then performability p_S of S if a function defined for each accomplishment level $a (a \in L)$ as:*

$$p_S(a) = P(\{\omega | Y_S(\omega) = a\})$$

Finally, we mention the traditional definition of performability (one possible concretization of the above framework) given in [193]. The accomplishment levels are modeled as system reward for dependable performance in the time interval $[t, t']$, where $t' >> t+\tau$. The reward rate during this interval is denoted by $r_s(g(x)), t \leq x \leq t'$, which is the performance at time x when the system's failure configuration is $g(x)$. Performability $Y_S(t, \tau)$ is defined as:

$$Y_S(t, \tau) = \frac{1}{t' - t} \int_t^{t'} r(g(x)) dx$$

In other words, performability is the integration of reward function. We will continue discussion about reward functions when we investigate Markov reward models (Section 3.1.7).

In the following chapter we will introduce models that enable us to perform system availability and performability analysis. We will see how, based on component parameters (e.g., MTTF, MTTR), it is possible to build a system model and to evaluate different indicators (e.g., reliability, availability, average downtime, performability etc.). Before we do that, however, let us investigate in more details service-oriented systems, which will be our systems-under-study in the remainder of this work, and how definitions we just introduced apply to them.

2.3 Services and Business Processes

Services are basic building blocks of service oriented architectures (SOA). SOA is an architectural attempt to describe and understand distributed systems which have minimal shared understanding among system components. As shown in Figure 2.5, roots of SOA can be traced back to distributed computing, programming languages and business computing.

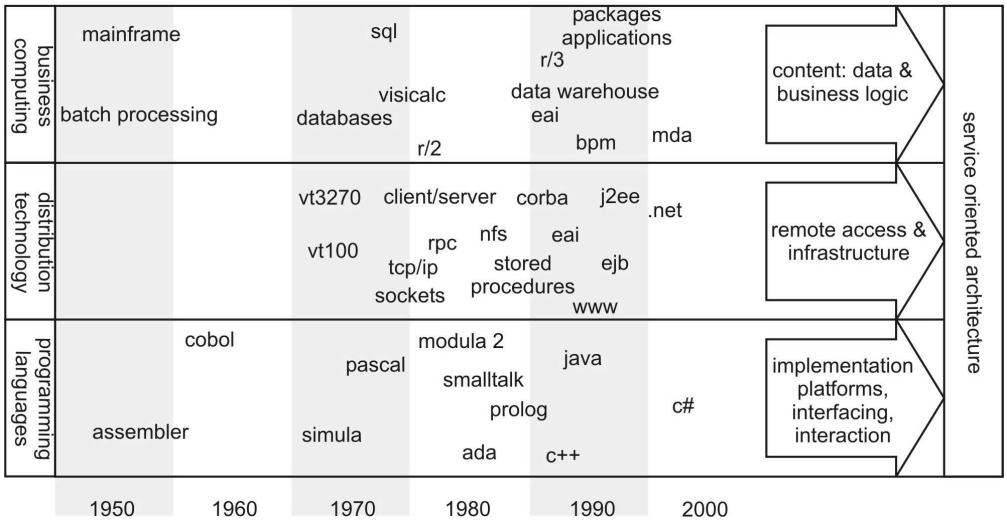

Figure 2.5: Historical perspective of the SOA development [118]

SOA is characterized with the following properties [94]:

- Logical view: The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.

- Message orientation: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA: using the SOA discipline one does not and should not need to

know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application that can be "wrapped" in message handling code that allows it to adhere to the formal service definition.

- Description orientation: A service is described by machine processable metadata. The description supports the public nature of the SOA: only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented by its description.
- Granularity: Services tend to use a small number of operations with relatively large and complex messages.
- Network orientation: Services tend to be oriented toward use over a network, though this is not an absolute requirement.
- Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

An important consequence of the above properties is profiling of the main SOA roles: requester, provider and broker (discovery agency). These roles and their interactions in SOA are shown in Figure 2.6.

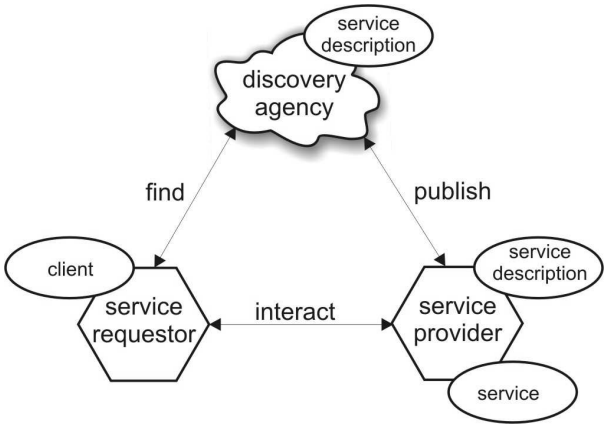

Figure 2.6: The SOA roles and interactions

The term service is heavily overloaded, and generally not exclusively used in the IT context. Merriam Webster Online [5] defines a service as

> a facility supplying some public demand that does not produce a tangible commodity.

The first part of this definition implies that a service can be performed by nearly anyone (e.g., a craftsman or an enterprise as well as hardware or software

systems), as long as service provider (supplier) and service consumer (public demand) are defined. The second part of the definition however, restricts certain activities, namely those that produce physical artifacts, from being considered a service.

The IT focus is more strict in the following definition, where a service is described as

> a meaningful activity that a computer program performs on request of another computer program. [118]

Similar is one of the earliest service definitions in the computational context where a service is

> a loosely-coupled computing task communicating over the internet, that plays the growing part in business-to-business interaction. [39]

The above definition does not specify that service consumers must necessarily be other services. Also, it clearly states that services focus to business-to-business communication, implicitly disqualifying services used internally, for example, in an enterprise, application or operating system.

Another technical-oriented definition states that a service is

> characterized by three parts: offered functionality, input and output messages, and interface address or port reference. [214]

Services can also be observed from a business perspective:

> A service captures functionality with a business value that is ready to be used. Services are made available by service providers. A services requires a service description that can be accessed and understood by potential service requestors. Software services are services that are realized by software systems. [217]

Apart from introducing the business aspect of service functionality, this definition is important because it also requires standardized and interpretable service description and allows services to be of a nature other than just software.

If we focus on IT-services, the predominant technology that is used today to implement them is the Web Service Architecture [9]. There are two accepted definitions of Web Services:

> A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards[1].

[1] http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

> A Web Service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web Service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.[2]

In [165] the following service types are distinguished: basic, intermediary, process-centric and enterprise services. Basic services are stateless and can be data- or logic-centric. The former handle persistent data (storage, retrieval, locking etc.) for one major business entity and provide strict interfaces to access its data. The latter encapsulate atomic implementation of business rules and processing (calculation). In practice there is a smooth transition between the two types as data-centric services may also use logic to check, validate or process data. Intermediate services are used to enforce micro consistency rules, performing gateway, adapter, façade and similar functions. They are implemented as simple business workflows (so called microflows) and are application-specific. More complex business tasks are realized using process-centric services which are often stateful. They call basic and intermediate services and thus encapsulate the knowledge and functionality of the organization's processes. Enterprise services are composed from process-centric services across company boundaries. Their interfaces have the granularity of business documents and are consequently very coarse grained. Their behavior is described by the service level agreements (SLA).

Based on previous considerations, in the remainder of this work we will be using the following definition of a service.

Def. 7 *A service is an abstraction of the infrastructure-, application- or business-level functionality. It consists of the contract, interface and implementation. Contract provides a description of constraints, functionality, purpose and usage of the service. Formal contract description can provide technology-independent abstraction suitable for verification and validation. Contract also imposes detailed semantics on functionality. Service interface provides means for clients to connect to the service, possibly but not mandatory via the network. Finally, business logic and business data are parts of service implementation that fulfills the service contract. Consequently, a service encapsulates business entity of a differing granularity and functional meaning.*

Contrary to the service definition, business process definition is fairly established and understood in the community. We will be using a definition aggregated from [208] and [217]. The relationship between business processes and services is given in Figure 2.7.

Def. 8 *A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other*

[2] http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/

organizations. A business process is represented by a business process model. It consists of a set of activities and execution constraints between them. A set of activities building a business process is organized in a workflow, therefore a business process is described using a workflow. Process activities are enacted by services, which are also elements of workflows. For expressing workflows, modeling languages such as Business Process Modeling Notation [157] are used.

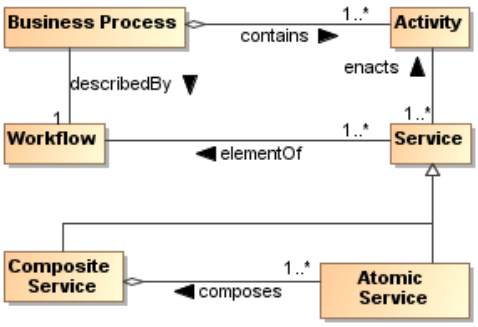

Figure 2.7: Services and business processes

2.4 Service Availability and Performability

Approaches adressing service availability almost exclusively investigate middleware concepts that can improve execution of the existing services. As such, they are closer to implementation of fault-tolerant systems (services), then reliablity and availability analysis, which is a necessary prerequisite for the fault-tolerant system design. However, in the majority of approaches, services are neither explicitly modeled nor is their availability and the improvement thereof quantified.

Service Availability Forum (www.saforum.org) is a consortium that develops high availability and management software interface specifications, specifically tailored to telecommunications sector. We are working closely with SAF to promote formal service availability modeling and quantification. In [40], a high availability middleware is presented which uses similar concepts. Many approaches introduce support for fault-tolerant execution of single or composite Web services, using concepts such as transactional message delivery or message bus to ensure reliability, as in [225, 77, 196, 226]. Other approaches use active and passive replication to ensure fault-tolerance [227], or semantic substitution [128], but also assess service availability using simplified experiments only. Other, non-standard mechanisms for achieving fault-tolerance, such as the use of generative communication and tuple space middleware, have been proposed [8]. Neither of these numerous approaches models reliability and availability explicitly or enables to quantify improvements gained using the proposed middleware concepts. Very often service availability itself is not even properly

defined and is perceived in its simplest from as the ratio of "correct" service invocations.

The Web Service Reliable Messaging specification (WS-R) [1] is used to guarantee SOAP message delivery for Web services architecture, but also does not provide a fault model or availability quantification. [57] extends this concept to reliable messaging at the business process level. Some works expand on the WS-ReliableMessaging to include stochastic models for calculating optimal restart time for improving service performance, again without generality or quantifiable (model-based) availability assessment [211].

Many QoS-aware frameworks for service composition claim to improve service availability such as [222, 223]. However, all QoS properties are usually informally defined, most frequently using simplified semantics of tuples (property,unit,value) and further treatment is reduced to measurements. In our opinion, all similar approaches suffer from the same downside: they attempt to fit all non-functional properties into one framework, disregarding their meaning, definition, modeling and potential conflicting requirements. Some approaches [96] even propose fault-prevention methods (e.g., self-healing) without defining a proper fault model. Contrary to all QoS-aware composition frameworks we focus on service reliability/availability only, but hope to treat it to significant and usable depth and level of details.

There are also experimental approaches to determine Web service availability using measurement based approaches, such as described in [169, 56, 183], including fault-injection based approaches as in [212, 76]. They all assume very simplistic fault-model and perform almost identical experimets of many thousand invocations to calculate uptime/downtime ratio for a given service.

A recent trend in the community can be observed, where dependability concepts from the embedded systems area such as [159] or [134] are being reused at the service level (e.g., see [89]). However, the work is mostly based on the analysis of error propagation and testability, and not on the rigorous availability assessment.

A notable exception from the trend of informal treatment of service availability is the work presented in [177, 178]. The approach develops a formal stochastic service availability model (Markov reward models) based on the assumption that failure of service resources causes failure of services and that there is abstract composite service description. Another approach is presented in [113] where availability is explicitly modeled for abstract services using combinatorial models. Several availability levels are calculated, such as user or service availability. However, important distinctions between our and these approaches are: both approaches require manual availability model generation, do not consider network, supporting infrastructure and people, do not support redundant resources, workflow descrpition is limited, and service models are required to have transition probabilities which is not realistic.

In [95] another semi-formal approach to modeling service availability is presented. It introduces formulas for calculating availability of sequence, parallel, choice and iteration compositions. Apart from some of the formulas being incorrect, the work defines service availability as a simple quotient of uptime and downtime, completely ignoring instantaneous, interval or user-perceived avail-

ability. Furthermore, it is not possible to model coverage, limited repair or priorities, as derived formulas are combinatorial in nature. Similarly, [224] introduces informal criteria for reliability of SOA systems, but they are intended for test-based validation only.

[219] presents an approach for assessing QoS parameters based on Bayesian networks. The basic idea is however that the network should observe the SLA fulfillment and learn if the user's requirements have been satisfied. To this end, values of QoS parameters are discretized and assigned levels such as high, moderate or low. This approach is neither able to accomodate for the stochastic nature of failures nor to quantify service availability.

Due to complexity, one of the main needs in service availability assessment is the ability to automatically generate availability model based on service and process description and the underlying infastructure. In [195] and [194] an approach for automatic availability model generation is presented which is limited to server boxes' configuration. Nevertheless, it is relevant as it enables abstract modeling of server configuration using predefined components, which are subsequently parameterized. Based on this information, availability model is generated. A slightly more general approach is presented in [24], enabling automatic state reduction of availability models, which however have to be entered manually at the first place.

The process management tools that we investigated (see Chapter 4 and Appendix B), such as IBM Tivoli, HP Mercury or Fujitsu Interstage, all offer different availability indicators, such as transaction throughput, number of concurrent clients, CPU/memory/disk usage. Based on these indicators, thresholds are defined and empirical availability estimation may be performed. However, even the most sofisticated tools lack support for direct availability monitoring – it has to be deduced from the aforementioned indicators. The strength of these tools however is in their capability to monitor the infrastructure and map services and processess to infrastructure elements. We also observed that unfortunately this mapping works best if the service ecosystem is based on or supported by the products (application servers, database servers etc.) of the same provider/company. Interesting work may be found in [44] which proposes a two step method for analyzing dependencies between business and IT services. In [97] the idea is further expanded to include determining impact and calculating cost of resource failures with respect to services and SLA. [213] introduces the notion of operator errors as one of the critical aspects, albeit in a very limited context (DBMS services).

The above examples indicate that, in order to understand service availability, an appropriate fault model has to be defined first. We argue that service availability must be perceived as the function of availability of the underlying ICT-layer, comprising of:

- hardware (e.g., servers, clients, workstations, clusters, grids)
- software (e.g., operating systems, database services, web services, custom applications, configuration)
- network (e.g., routers, switches, network cables, topology)

- supporting infrastructure (e.g., air-conditioning, power supply, physical security)
- people (e.g., users, administrators, maintenance and security personnel)

A successful approach to assess service and business process availability has to be able to determine functional dependency between the ICT-layer availability and the service availability. To define a comprehensive and consistent fault model, we introduce an additional ICT-layer to the Figure 2.7.

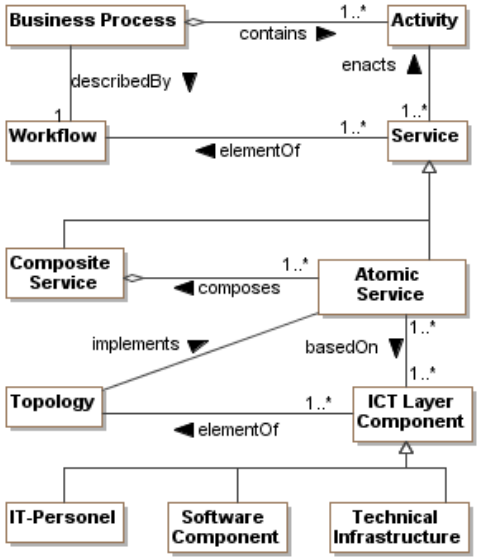

Figure 2.8: Extended architecture of services and business processes

The new layer comprises hardware, software, infrastructure and people components (Figure 2.8). They are organized into deployment topologies, which implement atomic services. Therefore, services are based on ICT-layer components. This will be the premise of our further investigation. We will now define service and business process availability.

The first step in this direction is to distinguish between faults, symptoms, errors and failures. The fault is incorrect state of a service, not necessarily leading to observable incorrectness or failure. The symptom is observable out-of-norm parameter behavior. The error is a manifestation of a fault (e.g., by observing symptoms) observed by a fault detector. Finally, we define the service and process failure:

Def. 9 *The failure is an event caused by errors which occurs when the service or business process deviates from the specified service or business process.*

The above definition assumes a perfect fault detector. However, if a fault detector is imperfect (has a coverage below 1) an undetected error may also cause a failure. In other words, faults which are root causes of undetected

errors can in principle cause failures directly. Thus, all failures and errors have fault root causes, but not all faults and not all errors must cause a failure.

As defined in [11], correct service, in the most general form, is delivered when the service implements the system function or process. A service failure is an event that occurs when the delivered service deviates from the correct service. A service can fail either because it does not comply with its functional specification, or because the specification did not adequately describe the system function required by service users. A service failure is transition from the correct to the incorrect service. The period of incorrect service is service outage. The transition from incorrect to correct function is service restoration. Deviation from correct service may assume different forms that are called service failure modes. In this context, availability can be simplified as the readiness for correct service.

In order to define a service fault model, we have to investigate four attributes that characterize incorrect services: failure domain, detectability of failures, consistency of failures and consequences of failures. The failure domain comprises content and timing failures. Failure domain is determined by parameters that modify attributes of QoS measures, as depicted in Figure 1.1. Other parameters, such as cost, transactions or authorization can be taken into consideration, however, we restrict discussion in this work to content and timing failures and do not discuss further QoS modifications such as price or even semantics. Time and content also subsume other modifications to some extent through MTTR and MTTF parameters. In the presence of content failures, the output information delivered by the service deviates from the correct output. Timing failures are characterized by the incorrect time offset of the service result delivery. Timing failures can be early and late. When combined, these two classes produce halting (potentially silent) failures and erratic failures. Failure detectability is related to the signaling of the service failure to the environment or to the user, and failures can be classified as signaled or unsignaled. Problematic issue is the user-perceived correctness, where the service may have been executed according to its specification and no failure signals will be emitted, however user is not satisfied with the results, as the service description did not correctly describe required functionality from the user's perspective. With respect to the failure consistency, service failures can be either consistent (incorrect service is perceived identically by all users, assuming there is more than one user), or inconsistent (where users perceive differently incorrect services, and some even receive correct service). Detectability and consistency of service failures will lead us to define user-perceived service availability later. Finally, with respect to consequence, service failures can be either minor (the harmful consequence of a failure is of a similar cost when compared with the benefit of a correct service) or catastrophic (cost of the harmful consequence is orders of magnitude higher than the benefit provided by correct delivery). Service failures, their domains, detectability, consistency and consequences are summarized in Figure 2.9.

Based on these preliminaries and the layered architecture from Figure 2.8, we define following service and business process failure modes.

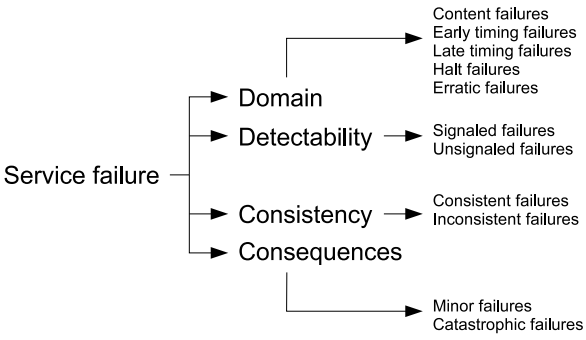

Figure 2.9: Service failure modes (figure adapted from [11])

Def. 10 *Temporal service failure mode describes failures which cause service to miss a deadline. A service will not respond in time if 1) a subset of ICT-components in a topology it is based on does not respond in time; 2) all ICT-components respond on time, but topology synchronization exceeds the deadline; 3) there is a deadlock.*

Def. 11 *Temporal business process failure mode describes failures which cause business process to miss a deadline. A business process will not respond in time if 1) a subset of services in a workflow does not respond in time; 2) all services respond in time, but workflow synchronization exceeds the deadline; 3) the workflow has a deadlock.*

Def. 12 *Value service failure mode describes failures which cause service to return incorrect value or perform the incorrect function. A service will respond with incorrect value if 1) a subset of ICT-components in a topology delivers incorrect values; 2) data and control flow in a topology are incorrect.*

Def. 13 *Value business process failure mode describes failures which cause business process to perform the incorrect business operation. A business process will perform the incorrect business operation if 1) a subset of services in a workflow delivers incorrect values; 2) workflow orchestration description is incorrect.*

Value failure mode is not restricted to pure functional correctness, but comprises non-functional properties expressed as service guarantees (or service level agreements) in the service contract. Thus a service may operate with reduced capacity or quality and still be considered correct. For more on non-functional capabilities and service contracts, see e.g., [146]. Based on the above failure modes, we define following types of service availability.

Def. 14 *Instantaneous service or business process availability is probability that service or business process is in the correct state and ready to perform its function at a specified time.*

Def. 15 *Interval availability is probability that service or business process is operating correctly during the given time interval. It is the expected proportion*

of time that service or business process is operational during the given time interval.

Def. 16 *Steady state availability is long-term probability that service or business process is available. It can be defined as the expected service or business process uptime over its lifetime.*

Finally, user percieved availability can be defined for an interval of for the system lifetime as follows:

Def. 17 *User-perceived interval availability is the number of correct service or business process invocations over the total number of invocations for a specified time interval, calculated (observed) for a particular service or a business process user.*

Def. 18 *User-perceived steady-state availability is the number of correct service or business process invocations over the total number of invocations estimated for the service or business process lifetime, calculated (observed) for a particular service or business process user.*

As already stated, changes in the environment may frequently cause an impact to sevices or business processes, causing one or more incorrect states (faults), without leading directly to failures. In such state, service may continue to offer some functionality, although it may not comply with its non-functional contract anymore, as it operates at a degraded performance level. Hence, to be able to define service and process performability, we first investigate what consists a service performance level.

Def. 19 *Service performance level is a value of QoS attribute, taken from a continuous or a discrete set, to which appropriate metric is assigned.*

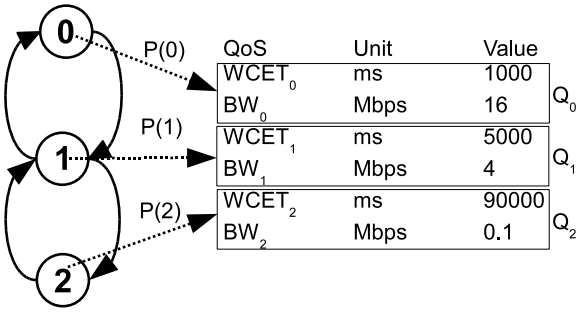

Figure 2.10: Service performability example

QoS attributes can be represented as triples, comprising type, unit and value (Figure 2.10), as suggested in [145]. An example QoS attribute may be worst case execution time, represented as $(WCET, 10, ms)$, meaning that the service guarantees an upper execution-time bound of 10 milliseconds. Using QoS attributes we define service performability.

Def. 20 *Service performability is function $P(s)$ that maps the current service state $s \in S$ to the set of QoS attribute values Q, $P : S \rightarrow Q$.*

In other words, performability is the quantifiable and predictable ability of a service or business process to respond to faults by assuming one of the performance levels without entering the (total) failure state. For example, a service that provides video streaming may respond to faults in the network infrastructure by lowering bandwidth and increasing response time. Figure 2.10 shows a service with three states (0, 1 and 2) to which three QoS levels are associated. Similarly, if a fault in the server infrastructure of a financial service is encountered, the transaction throughput and number of concurrent users can be lowered, or the execution and processing time can be extended.

Chapter 3

Availability and Performability Models

3.1 Analytical Models

The overall system availability assessment can usually be made easier by breaking up the system into components, which may or may not be statistically independent. In the following sections we will investigate the most frequently used models for partitioning a system into components or states, and for calculating overall system availability, assuming that availability properties of the unit components are known (obtained e.g., from historical data, component specification sheets, standard catalogs or derived using qualitative methods described in the section 3.2). We will investigate combinatorial models (reliability block diagrams, fault trees and reliability graphs), Markov and semi-Markov models, Petri nets and stochastic activity networks.

3.1.1 Reliability Block Diagrams

Reliability block diagrams (RBD) are used to represent the logical structure of a system, with respect to how reliability of its components affects overall system reliability and availability. Basic configurations in which components of a diagram may be combined are series, parallel and k-out-of-n configurations. We further assume that components are independent, that is, failure information for one component provides no information about (i.e. does not affect) other components in the system. There are extensions to this basic model, where dependency can be modeled with configurations such as load sharing or standby, but they are out of the scope of this manuscript.

In the series configuration, a failure of any component results in a failure of the entire system. In other words, for the whole system to work, all serial components must also work. For a series configuration, the distribution function for the failure time of a system with N components is given by:

$$F_s(t) = 1 - \prod_{i=1}^{N}(1 - F_i(t))$$

F_i is the distribution function for the failure time of the i-th component. Direct consequence is that the system reliability is equal to the product of reliabilities of its constituent components:

$$R_s = \prod_{i=1}^{N} R_i$$

The first equation can be proven using convolution, but the second one has a more intuitive explanation. Reliability of a series configuration is the probability that all components are functioning:

$$R_s = P(X_1 \cap X_2 \cap ... \cap X_{n-1})$$

where X_i is the event of component i in a series being operational. Using conditional probabilities, this equation is equivalent to:

$$R_s = P(X_1)P(X_2|X_1)P(X_3|X_1X_2)...P(X_n|X_1X_2...X_n)$$

In case of independent components, this equation becomes:

$$R_s = \prod_{i=1}^{N} P(X_i)$$

which shows that reliability of a series configuration of independent elements with equivalent distributions is the product of components' reliabilities. Similarly, it can also be shown that following holds for the system availability:

$$A_s = \prod_{i=1}^{N} A_i$$

We assumed here that the failure time and the repair time distributions are independent. That means that the system has enough resources to repair all components at the same time, if necessary. Consequently, overall system failure rate can be calculated as:

$$\lambda_s = \sum_{i=1}^{N} \lambda_i$$

Therefore, system MTTF can be calculated as:

$$MTTF_s = \frac{1}{\sum_{i=1}^{N} \frac{1}{MTTF_i}}$$

In a parallel configuration, at least one of components must succeed for the system to succeed. In other words, the system will function if any one (or more) of the components is working. The components in parallel are also referred to as the redundant units. It can be shown that the following holds for the distribution function for the failure time of a parallel configuration of N components:

$$F_p(t) = \prod_{i=1}^{N} F_i(t)$$

It follows that the system reliability is:

$$R_p = 1 - \prod_{i=1}^{N} (1 - R_i)$$

In order for a redundant system to fail, all redundant units must fail. Therefore, probability of a system failure is the probability that all components fail:

$$Q_p = P(X_1 \cap X_2 \cap ... \cap X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1X_2)...P(X_n|X_1...X_{n-1})$$

where X_i is the event of failure of the component i and Q_p is unavailability of the parallel configuration. If components are independent, this equation becomes:

$$Q_p = \prod_{i=1}^{N} Q_i$$

Finally, system reliability is:

$$R_p = 1 - Q_p = 1 - \prod_{i=1}^{N} (1 - R_i)$$

Similarly, it can be shown that availability of the parallel configuration is given by:

$$A_p = 1 - \prod_{i=1}^{N} (1 - A_i)$$

The system failure rate and MTTF of the parallel configuration are more difficult to derive in a closed form. In case of a non-repairable system, integration of the composite reliability function for the parallel configuration yields following expression:

$$MTTF_p = \sum_{i=1}^{n} \frac{1}{\lambda_i} - \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{\lambda_i + \lambda_j} + \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n} \frac{1}{\lambda_i + \lambda_j + \lambda_k} - ...+$$

$$+(-1)^{n+1} \frac{1}{\sum_{i=1}^{n} \lambda_i}$$

The complete integration can be found in [75]. Assuming that all components have equal failure rates λ, the expression is simplified into a well-known formula:

$$MTTF_p = \frac{1}{\lambda}\Big[1 + \frac{1}{2} + ... + \frac{1}{n}\Big]$$

In case of a repairable system, problem is more complex as the failed component may be repaired, which influences overall system failure rate. The system failure rate is obviously a function of components' repair rates. It can be shown that following holds for the repair rates of a parallel configuration:

$$MTTR_p = \frac{1}{\sum_{i=1}^{n} \frac{1}{MTTR_i}}$$

Knowing system availability, mean time to failure can now be expressed as a function of the system availability and mean time to repair:

$$MTTF_p = \frac{A_p}{1 - A_p} MTTR_p$$

The third basic configuration is k-out-of-n, and it includes both series and parallel configurations as special cases. This type of configuration requires that at least k components succeed out of total n parallel components for the system to succeed. When $k = n$, we have n-out-of-n which is equivalent to the series configuration, and when $k = 1$, we have 1-out-of-n, which is equivalent to the parallel configuration. The distribution function for the failure time of identically distributed k-out-of-n configuration is:

$$F_{k|n}(t) = \sum_{i=k}^{n} \binom{n}{i} F(t)^i (1 - F(t))^{n-i}$$

Let us assume that components are independent and identical (they all have the same MTTF and MTTR). The system reliability is:

$$R_{k|n} = \sum_{i=k}^{n} \binom{n}{i} R^i (1 - R)^{n-i}$$

MTTR and MTTF are in this case given by:

$$MTTR_{k|n} = \frac{MTTR}{n - k + 1}$$

$$MTTF_{k|n} = MTTF\Big(\frac{MTTF}{MTTR}\Big)^{n-k} \frac{(n-k)!(k-1)!}{n!}$$

Finally, availability of the identical k-out-of-n configuration can be calculated as:

$$A_{k|n} = \frac{MTTF_{k|n}}{MTTF_{k|n} + MTTR_{k|n}}$$

The following illustrative example will demonstrate how to use the RBD methodology to model a fault-tolerant system and to evaluate overall system parameters (availability, MTTF and MTTR). Let us observe a fault-tolerant computer system, comprising of single units for the CPU, memory and console,

redundant (parallel) units for the power supply and 3-out-of-4 redundant disk units. Reliability block diagram describing such a system is given in Figure 3.1, with MTTF and MTTR values of all components (exponential distribution is assumed).

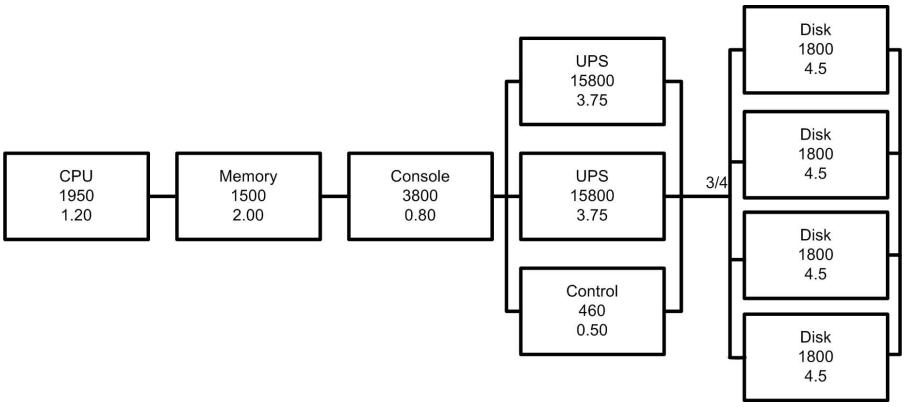

Figure 3.1: Reliability block diagram of a fault-tolerant computer architecture

We will now calculate the MTTF, MTTR and availability of the entire system. The failure rate of series configuration (CPU, memory and console) is:

$$\lambda_1 = \lambda_{CPU} + \lambda_{MEM} + \lambda_{CON}$$

The MTTF of this configuration is:

$$MTTF_1 = \frac{1}{\lambda_1} = \frac{1}{\lambda_{CPU} + \lambda_{MEM} + \lambda_{CON}} = \frac{1}{\frac{1}{MTTF_{CPU}} + \frac{1}{MTTF_{MEM}} + \frac{1}{MTTF_{CON}}}$$

Substituting values for the CPU, memory and console, and assuming that MTTRs are given in hours, we obtain:

$$MTTF_1 = 693.2 \ hours$$

Availability of the serial system is equivalent to the product of component availabilities:

$$A_1 = A_{CPU} \cdot A_{MEM} \cdot A_{CON} =$$

$$= \frac{MTTF_{CPU}}{MTTF_{CPU} + MTTR_{CPU}} \cdot \frac{MTTF_{MEM}}{MTTF_{MEM} + MTTR_{MEM}} \cdot$$

$$\cdot \frac{MTTF_{CON}}{MTTF_{CON} + MTTR_{CON}} = 0.99784418$$

Finally, $MTTR_1$ is easily calculated as:

$$MTTR_1 = \frac{1 - A_1}{A_1} MTTF_1 = 1.4976 \ hours$$

Let us observe the second subsystem, comprising three parallel components: two redundant uninterruptable power supplies and a control unit. Availability of this parallel configuration is:

$$A_2 = 1 - (1 - A_{UPS})(1 - A_{UPS})(1 - A_{CON}) =$$
$$= 1 - \Big(1 - \frac{MTTF_{UPS}}{MTTF_{UPS} + MTTR_{UPS}}\Big)^2 (1 - \frac{MTTF_{CON}}{MTTF_{CON} + MTTR_{CON}})$$

Substituting values we obtain the availability:

$$A_2 = 0.99999999994$$

Mean time to repair of this configuration is:

$$MTTR_2 = \frac{1}{\frac{1}{MTTR_{UPS}} + \frac{1}{MTTR_{UPS}} + \frac{1}{MTTR_{CON}}} = 0.3947368 \ hours$$

Knowing the availability and the mean time to repair, the mean time to failure of this configuration is:

$$MTTF_2 = \Big(\frac{A_2}{1 - A_2}\Big) MTTR_2 = 6,456,914,809 \ hours$$

The third subsystem is 3-out-of-4 configuration of identical disk drive components, meaning that at most one drive may fail without causing the system failure. The MTTR of this configuration is:

$$MTTR_3 = \frac{MTTR}{n - k + 1} = \frac{MTTR}{2} = 2.25 \ hours$$

The MTTF is:

$$MTTF_3 = MTTF\Big(\frac{MTTF}{MTTR}\Big)^{n-k} \frac{(n-k)!(k-1)!}{n!} = \frac{1}{12}\frac{MTTF^2}{MTTR} = 60000 \ hours$$

The availability of the configuration is then calculated as:

$$A_3 = \frac{MTTF_3}{MTTF_3 + MTTR_3} = 0.9999625$$

Based on the rules for series configuration, overall availability, system mean time to failure and mean time to repair are:

$$A_s = A_1 \cdot A_2 \cdot A_3 = 0.9978033$$

$$MTTF_s = \frac{1}{\frac{1}{MTTF_1} + \frac{1}{MTTF_2} + \frac{1}{MTTF_3}} = 685.25 \ hours$$

$$MTTR_s = \frac{1 - A_s}{A_s} MTTF_s = 1.51 \ hours$$

	Availability	MTTF	MTTR
CPU, MEM, CON	0.9978441823840	693.1711880262	1.497579165575
Power	0.9999999999389	6456914847	0.3947369468251
Disk	0.9999628107504	60000	2.25
Overall system	0.9978067645694	685.2544702083	1.506227895453

Table 3.1: RBD model evaluation with SHARPE

Solving the model in SHARPE tool (see Section B.1.36 in the next chapter for detailed SHARPE description), we obtain results given in Table 3.1, which confirm manual calculations.

After this example, one may tend to think that RBD model can be solved manually in any general case. We solved the simplest RBD variant, with exponential distributions for all components which are independent and arranged using the combination of series, parallel and k-out-of-n configurations. In more complex cases, where components depend on each other, use different (complex, non-exponential) distributions or configurations (e.g., load sharing, standby, multiblocks, inherited blocks, mirrored blocks etc.), or have limited repair capacity, tool and simulation support becomes necessary, as manual solving becomes practically impossible.

3.1.2 Fault Trees

Fault trees, similar to reliability block diagrams, represent all sequences of individual component failures that may cause the system to stop functioning. Fault trees apply deductive logic to produce a fault-oriented pictorial diagram which allows one to analyze system safety and reliability. The main difference between fault trees and RBDs is that with RBD one is working in the "success space", that is, looking at the success combinations of system components. Contrary, a fault tree diagram is generated in the "failure space", identifying all possible system failure combinations. Hence, fault trees are also a design aid for identifying general fault classes of a fault tolerant system.

The starting point of every fault tree is definition of a single, well-defined and undesirable event (e.g., a system failure), which represents the root of a tree. The tree is then built in a top-down manner, combining logic gates (such as AND and OR), and events. Each gate may have other gates or events as input. The reduction process stops when we reach basic events which we do not want to reduce further. Probability of higher level events can be calculated by combining probabilities of lower level events. We will further assume that basic events are mutually independent.

Let us consider basic gate types that can be used in a fault tree. In an OR gate, the output event occurs if at least one of input events occurs. In system reliability terms, this implies that if any component fails (input) then the system will also fail (output). When using RBDs, the equivalent is a series configuration. Failure time, reliability and availability of an OR gate with n inputs are:

$$F_{OR}(t) = 1 - \prod_{i=1}^{n}(1 - F_i(t))$$

$$R_{OR} = \prod_{i=1}^{n} R_i$$

$$A_{OR} = \prod_{i=1}^{n} A_i$$

In an AND gate, the output event occurs if all input events occur. In system reliability terms, this implies that all components must fail (input) in order for the system to fail (output). When using RBDs, the equivalent is a simple parallel configuration. Failure time, reliability and availability of an AND gate with n inputs are:

$$F_{AND}(t) = \prod_{i=1}^{n} F_i(t)$$

$$R_{AND} = 1 - \prod_{i=1}^{n}(1 - R_i)$$

$$A_{AND} = 1 - \prod_{i=1}^{n}(1 - A_i)$$

The last basic fault tree gate is k-voting OR gate. In a k-voting OR gate, the output event occurs if k or more of input events occur. In system reliability terms, this implies that if any k or more components fail (input) then the system will fail (output). The equivalent RBD construct is a k-out-of-n parallel configuration with a distinct difference. In general case, the k-voting OR gate with n inputs is *not equivalent* to the k-out-of-n parallel configuration. The reason is different space in which two diagram types are constructed. Let us compare 3-out-of-4 RBD configuration, which has the following meaning: at most one component may fail, that is, three components are required to keep the system running. If we replace this component with a 3-voting OR gate with four inputs, we will not obtain the equivalent configuration. This gate means that if and only if three out of four elements fail, the gate (system) will fail, which is not equivalent with the RBD. The equivalent of 3-out-of-4 RBD is 2-voting OR gate, because it fails if two systems fail, meaning that at most one system may fail in order for a system to continue to function. Failure time, reliability and availability of an k-voting OR gate with n identically distributed inputs are:

$$F_{K|N}(t) = \sum_{i=k}^{n} \binom{n}{i} F(t)^i (1 - F(t))^{n-i}$$

$$R_{K|N} = \sum_{i=n-k+1}^{n} \binom{n}{i} R^i (1 - R)^{n-i}$$

$$A_{K|N} = \frac{MTTF_{K|N}}{MTTF_{K|N} + MTTR_{K|N}}$$

To illustrate fault tree application, we will model the scenario from previous section using the fault tree methodology. Resulting diagram is shown in Figure 3.2.

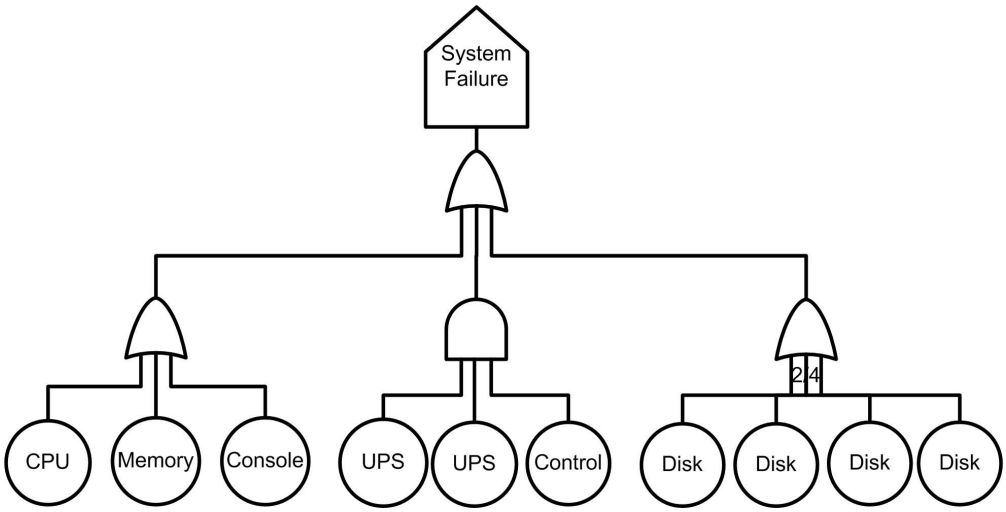

Figure 3.2: Fault tree diagram of a fault-tolerant computer system

Finally, we will verify results from the previous section by solving the fault tree in IsoGraph FaultTree+ tool (see section B.1.15 in the next chapter for more information on this software package). Analysis results are given in Table 3.2.

	Availability	MTTF	MTTR
CPU, MEM, CON	0.99784418	692.883	1.49758
Power	0.999999999938866	6453350000	0.394737
Disk	0.9999628108	60575.6	2.25375
Overall system	0,99780707	685.154	1.50619

Table 3.2: Fault tree model evaluation with IsoGraph

Apart from three basic gates, there are additional gates which can be used to build more complex configurations. The inhibit gate generates a fault if all input events occur and an additional conditional event occurs. The priority AND gate generates a fault if all input events occur in a specific sequence. The dependency AND gate generates a fault if all input events occur, however input events are dependent, that is, the occurrence of each event affects the probability of other events' occurrence. The exclusive OR gate (or XOR) generates a fault if exactly one input event occurs. For simple configurations, a bijective mapping between RBDs and fault trees exist. With some specific exceptions, it is also generally possible to map any fault tree into equivalent RBD. The opposite does

not hold. Conversion rules between RBDs and fault trees are given in Table 3.3. Note that, apart from three basic gates/configurations, other represent tool-specific extensions and may not be fully supported by all tools.

RBD configuration	Fault tree gate
Simple parallel	AND
Series	OR
k-out-of-n parallel	(n-k+1)-voting
Simple parallel with condition	Inhibit
Standby parallel	Priority AND
Load sharing parallel	Dependency AND
N/A	XOR

Table 3.3: Conversion between RBDs and fault trees

Note also that the XOR gate has no equivalent RBD configuration, as it would imply that two-component configuration would work even if both systems fail.

3.1.3 Reliability Graphs and Complex Configurations

The reliability graph model consists of a set of nodes and edges, where edges can represent either components that can fail or structural relationships between components. The graph also has two special nodes, the source with no incoming edges and the sink, with no outgoing edges. The semantics of a reliability graph is that a system thus described will fail if and only if there is no path from the source to the sink. The main difference between basic reliability block diagrams and reliability graphs is that failure probabilities, failure rates or unavailability values or functions are assigned to edges, and not to nodes. Furthermore, reliability graphs support complex configurations where multiple paths exist between two nodes.

Reliability graphs are analyzed using a factoring algorithm. An edge is chosen as the pivot, and two graphs are constructed: *up* and *down*. The *up* graph represents an equivalent graph where the pivot edge does not fail and the *down* graph is an equivalent graph where the pivot edge fails. The equivalent graph is constructed by removing pivot edge, irrelevant nodes and edges (down case) and combining edges in series (up case). This procedure is continued for all generated graphs until only basic series-parallel configurations remain, which are solved using well known equations for distribution functions:

$$F(t) = \begin{cases} 1 - \prod_{i=1}^{n}(1 - F_i(t)) & \text{for a series configuration} \\ \prod_{i=1}^{n} F_i(t) & \text{for a parallel configuration} \end{cases}$$

The overall system distribution is then calculated using the total probability formula. Let us demonstrate factoring of a reliability graph and conversion to RBDs using several examples. The reliability graph and its first factorization are given in Figure 3.3.

The edge A has been chosen for factoring. When A is down, nodes 1, 2 and 3 are unreachable, therefore edges B, C and I are also eliminated from the

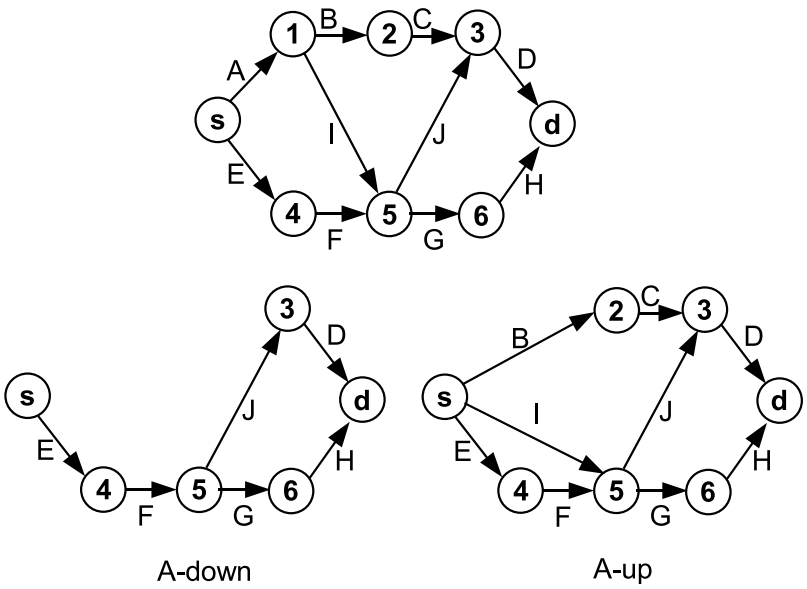

Figure 3.3: Reliability graph and its factorization

$A - down$ graph. When A is up, serial connection is established between the source node and node 2 (edge B), as well as between the source node and node 5 (edge I) in the $A - up$ graph. The $A - down$ graph is simple series-parallel graph, for which we can calculate the distribution function (assuming that edge i is described with distribution function F_i):

$$F_{A-down} = 1-(1-F_E)(1-F_F)(1-(1-(1-F_J)(1-F_D))(1-(1-F_G)(1-F_H)))$$

The $A - up$ graph, however, still contains complex paths that cannot be evaluated. Therefore, the factoring procedure is repeated, using edge D as a pivot. In the $A - up, D - down$ graph, edges B, C and J are removed as the consequence of a downed edge D and unreachability of the nodes 2 and 3. In the $A - up, D - up$ graph, serial connection is added between nodes 2 and d (edge C) and 5 and d (edge J). The result of this factoring step is shown in Figure 3.4.

Now both graphs $A - up, D - down$ and $A - up, D - up$ are simple series-parallel graphs for which distribution functions $F_{A-up,D-down}$ and $F_{A-up,D-up}$ can be found analogously to previous example of the F_{A-down}. The overall system distribution function is obtained by applying the total probability formula, which states that if $\{S_1, ..., S_n\}$ is a finite or countably infinite partition of a probability space and each set S_n is measurable, then for any event X the following holds:

$$P(X) = \sum_{i=1}^{n} P(S_i)P(A|S_i)$$

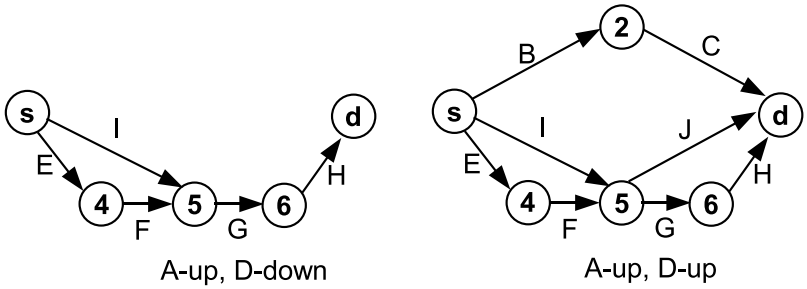

Figure 3.4: Further factoring of reliability graph from Figure 3.3

Taking into account that $R(t) = 1 - F(t)$, this translates into:

$$F = F_A F_{A-down} + (1 - F_A) F_D F_{A-up,D-down} + (1 - F_A)(1 - F_D) F_{A-up,D-up}$$

Let us observe another example of a fault tolerant computer system with two memory modules, two cache modules and one controller. The system fails if and only if both memory modules or both cache modules fail. Reliability graph and the equivalent reliability block diagram of this system are shown in Figure 3.5.

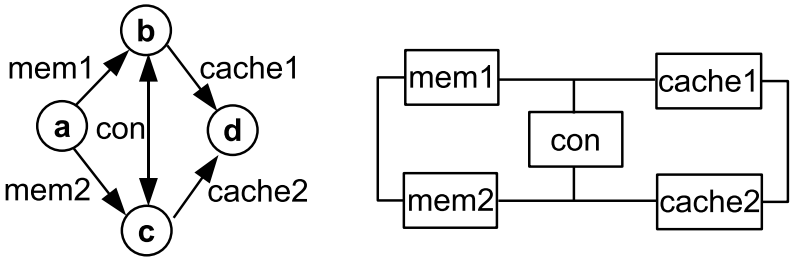

Figure 3.5: Reliability graph and its equivalent RBD

This particular RBD configuration is called a bridge, and cannot be solved using rules for series-parallel configurations. There are several methods for solving such complex configurations, among others decomposition, event space and path-tracing method. Pivoting procedure for reliability graphs is equivalent to decomposition procedure for RBDs. Choosing the controller component as pivot/decomposition component, we obtain factoring/decomposition shown in Figure 3.6.

The overall system availability is calculated using total probability formula:

$$A_S = A_{CON} \cdot (A_S|CON) + (1 - A_{CON}) \cdot (A_S|\overline{CON})$$

Results of manual RBD calculation are given in Table 3.4, together with the solution SHARPE gives for the equivalent reliability graph from Figure 3.5. Exponential distribution and following parameters were assumed for the components: memory MTTF=1500, MTTR=2; controller MTTF=1000, MTTR=3;

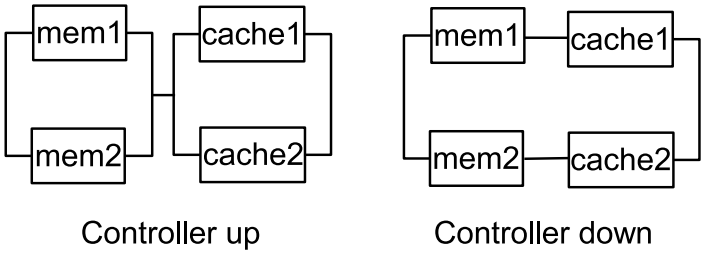

Figure 3.6: Decomposition of RBD from Figure 3.6

cache MTTF=3000, MTTR=0.5. Note that SHARPE result confirms the result obtained by applying decomposition method and total probability formula.

$A_S \mid CON$	0,9999981991850
$A_S \mid \overline{CON}$	0,9999977560711
A_{CON}	0,9970089730808
$1 - A_{CON}$	0,0029910269192
A_S manual	0,9999981978597
A_S SHARPE	0,99999819785963439682830312

Table 3.4: Results of the reliability graph/RBD analysis

Finally, let us examine another complex model: fault tree with repeated events. Consider the following example of a computer system with two processors (P1 and P2) and three memory modules, two of which are private (exclusive for each processor, M1 and M2), and one is shared (M3). The systems does not fail as long as one processor is up and can access either its private memory block or the shared memory. The fault tree model of this system is given in the left part of the Figure 3.7.

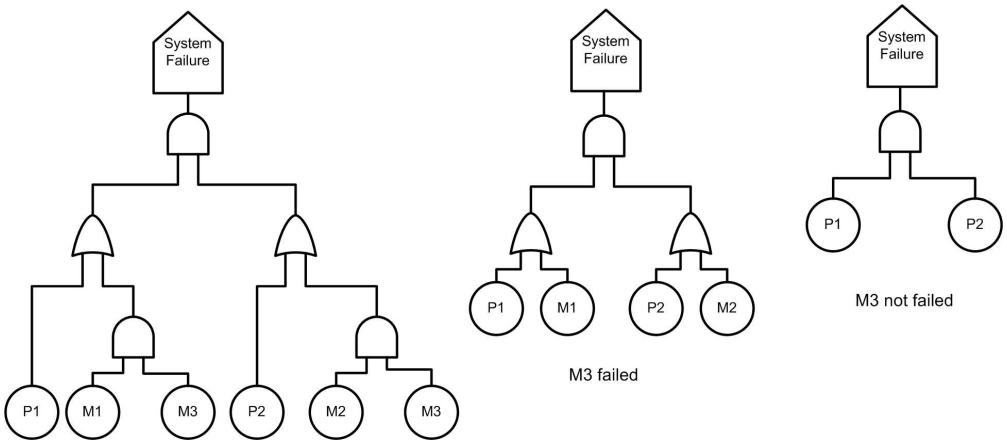

Figure 3.7: Fault tree with repeated events and its decomposition

This fault tree cannot be evaluated using expressions from Section 3.1.2.

Instead, decomposition is performed using the shared event (in this case failure of the shared memory module $M3$ as the pivotal element). When $M3$ fails, then the failure of $M1 \wedge M3$ and $M2 \wedge M3$ depends only on $M1$ and $M2$ respectively. Therefore, decomposition for this case is:

$$Failure|\overline{M3} = (P1 \vee M1) \wedge (P2 \vee M2)$$

When M3 is up, both *and* gates become true, therefore, only the following expression remains:

$$Failure|M3 = P1 \wedge P2$$

Both decompositions are shown in the right part of Figure 3.7. Decomposed fault trees can now be solved using equations from Section 3.1.2, and the overall system distribution is calculated using total probability formula, as already demonstrated. Reliability graph equivalent to the fault tree model with repeated events is given in Figure 3.8.

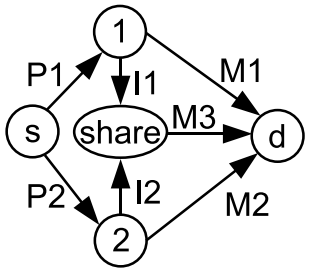

Figure 3.8: Reliability graph equivalent to the fault tree from Figure 3.7

Edges $P1$ and $P2$ represent processor failures and edges $M1$, $M2$ and $M3$ represent memory failures. In this graph another edge type is introduced: structural relationship edge ($I1$ and $I2$). These edges do not represent physical system components, instead they are used to provide shared semantics for edge $M3$, enabling each processor to access the shared memory. Therefore, $I1$ and $I2$ cannot fail, and are hence assigned infinite distributions $F_{I1} = F_{I2} = 0$. The graph can be analyzed using usual factoring procedure.

Note on solution algorithms. Three models described up to now (RDB, fault trees and reliability graphs) are related. A fault tree without repeated events is equivalent to RDB, and both are subsets of reliability graphs, which are in turn subset of fault trees with repeated events. Although all models can be in principle solved using the methods we described, there are more efficient ways. Most of the tools rarely use direct solution methods (reduction and series-parallel equations), but rather rely on generating minimal paths through the structure graph (e.g., a fault tree). Functioning of these algorithms can be briefly described in two steps. In the first step, set of minimal paths is generated. A path is defined as a set of components (e.g., events or edges) for which the system is up if all components are up. A minimal path is a

path which has no proper subpaths. The probability of a minimal path is obtained by multiplication of component probabilities. This is possible due to the assumption of independent component failures. In step two, system probability (reliability) is determined. If all minimal path are disjoint, the resulting reliability is multiplication of minipath reliabilities. However, this is rarely the case, therefore minipaths have to be combined. Two frequent algorithms are inclusion-exclusion [148] and sums of disjoint products [168], whose description is outside of scope of this manuscript.

3.1.4 Markov Models

Combinatorial models, such as reliability block diagrams, fault trees or reliability graphs, assume stochastic independence between components: the failure or repair of a component is not affected by other components. If there is a need to model more complex interactions, where the failure of a component influences behavior of other components, other kinds of models must be used. An example of such scenario is load-balancer with four active instances. If one instance fails, remaining three will have higher load which will likely impact their reliability. This requirement cannot be modeled using combinatorial approaches described so far. One possibility that can be used to cover this class of problems are Markov models.

Markov models are based on stochastic processes. A stochastic process is a family of random variables $X(t)$ defined on a sample space. The sample space is a set of all possible evolutions of the states. Values assumed by $X(t)$ are states, and the set of all possible states is the state space. When the value of $X(t)$ changes, process has performed a state transition. The state space can be either discrete or continuous. If the state space is discrete, the process is called a chain. We will discuss only stochastic processes with discrete state space, that is, chains. The time parameter of a stochastic process can also be discrete or continuous and we will cover both continuous-time chain and discrete-time chain cases.

Homogeneous Discrete-state Markov Process

Markov chain is a discrete-state stochastic process where the current state depends only on the immediately preceding state. We will define Markov chains in a formal way now. Let $X(t)$ be a discrete-state stochastic process and $P(X(t_n) = j)$ be the probability that the process is in state j at time t_n. $X(t)$ is a Markov chain if, for any ordered time sequence $t_0 < t_1 < ... < t_n$, the conditional probability of the process being in any state j at time t_n is given as:

$$P[X(t_n) = j | X(t_{n-1}) = i_{n-1}, X(t_{n-2}) = i_{n-2}, ..., X(t_0) = i_0] =$$

$$= P[X(t_n) = j | X(t_{n-1}) = i_{n-1}]$$

This equation defines that the state of a Markov chain after a transition may depend on the state immediately before it, but it cannot depend on any

other states before that. The entire past history is always summarized by the current state. If the conditional probability from previous equation is invariant with respect to the time origin t_n, Markov chain is homogeneous:

$$P[X(t) = j|X(t_n) = i_n] = P[X(t - t_n) = j|X(0) = i_n]$$

We will be working with homogeneous chains. This will enable us to disregard the time that a system spends in the current state. If we choose t_n such that the process was already in state j for some time, the probability of finding the system in some other state at time $t > t_n$ depends on state j, but not on how long the system was in it. The system is therefore memoryless as it disregards all previous states as well as the time spent in the current state.

The Markov chain analysis gives the state probabilities for finite values of t and for $t \to \infty$. The probability $\pi_j(t)$ of state j is the probability that the process is in state j at time t:

$$\pi_j(t) = P(X(t) = j)$$

The vector of all state probabilities is $\pi(t) = [\pi_0(t), \pi_1(t)...]$. The transition probability $P_{ij}(t - u)$ is the probability that the system is in state j at time t given that it was in state i at time u:

$$P_{ij}(t - u) = P[X(t) = j|X(u) = i]$$

Often, $u = 0$ is selected. The matrix $\mathbf{P}(t)$ is the square matrix of transition probabilities $P_{ij}(t)$. The Chapman-Kolmogorov equation states that, for a system to be in state j at some time t, it must have been in some state i at time u and must have made zero or more transitions to reach j:

$$\pi(t) = \pi(u) \cdot \mathbf{P}(t - u)$$

This is the basic equation we will be using to calculate probabilities $\pi(t)$. We will now define discrete-time Markov chains (DTMC) and continuous-time Markov chains (CTMC).

In case of a DTMC, let us define the points where state changes as $\{0, 1, ...\}$. Furthermore, let us define $\mathbf{P} = \mathbf{P}(1)$ and let p_{ij} be the (i, j)-th element of $\mathbf{P}$. The p_{ij} is one-step transition probability then. The matrix $\mathbf{P}$ has the following properties: all elements are in the range $[0, 1]$ and the sum of all elements in one row must be equal to one. Such a matrix is also called a stochastic matrix. For a homogeneous DTMC, Chapman-Kolmogorov equation becomes:

$$\pi(n + 1) = \pi(n) \cdot \mathbf{P}$$

Based on this, state probabilities $\pi(n)$ can be computed in terms of the initial probability vector $\pi(0)$:

$$\pi(n) = \pi(0) \cdot \mathbf{P}$$

If $\lim_{n\to\infty} \pi(n)$ exists, then the following system of equations can be used to calculate the limiting (steady-state) probability vector:

$$\pi = \pi \cdot \mathbf{P}$$

$$\pi \cdot \mathbf{e} = 1$$

where $\mathbf{e} = [1, 1..., 1]^T$.

Let us derive the equivalent system for CTMC. If we substitute $u = t - \Delta t$ and subtract $\pi(t - \Delta t)$ from both sides of the Chapman-Kolmogorov equation, we get:

$$\pi(t) - \pi(t - \Delta t) = \pi(t - \Delta t)[\mathbf{P}(\Delta t) - \mathbf{I}]$$

Dividing the equation with Δt and taking the limit when $\Delta t \to 0$, the following is obtained:

$$\frac{d\pi(t)}{dt} = \pi(t) \lim_{\Delta t \to 0} \frac{\mathbf{P}(\Delta t) - \mathbf{I}}{\Delta t}$$

If we define $\delta_{ij} = 1$ if $i = j$ and zero otherwise, and define the matrix $\mathbf{Q}$ as:

$$q_{ij} = \lim_{\Delta t \to 0} \frac{P_{ij}(\Delta t) - \delta_{ij}}{\Delta t}$$

The previous equation can be rewritten as:

$$\frac{d\pi(t)}{dt} = \pi(t)\mathbf{Q}$$

This is Kolmogorov differential equation, and the matrix $\mathbf{Q}$ is called the generator matrix of the CTMC. If $\lim_{t \to \infty} \pi(t)$ exists, the following system of linear equations can be used to calculate limiting (steady-state) probabilities:

$$\pi \cdot \mathbf{Q} = 0$$

$$\pi \cdot \mathbf{e} = 1$$

The scalar form of these equations is:

$$\pi_i q_i = \sum_{j \neq i} \pi_j q_{ji}$$

This formula is known as steady-state balance equation for state i and it states that the rate of flow into state i equals the rate of flow out of state i in the steady-state. Furthermore, the following also holds:

$$\sum_i \pi_i = 1$$

If the interval Δt is small, then:

$$1 - P_{ii}(\Delta t) = -q_{ii}\Delta(t)$$

$$P_{ij}(\Delta t) = q_{ij}\Delta t, i \neq j$$

Since

$$\sum_{j \neq i} P_{ij}(\Delta t) + P_{ii}(\Delta t) = 1$$

the following holds if $\Delta t \to 0$:

$$\sum_{j} q_{ij} = 0$$

Furthermore, the diagonal element equals the negative sum of the off-diagonal elements:

$$q_{ii} = -\sum_{j \neq i} q_{ij}$$

These observations will help us in constructing generator matrices. Now we will discuss the analysis of Markov chains, and then show how to use them to model reliability and availability.

Analysis of Discrete Time Markov Chains

Let us construct a DTMC which models a modern multithreaded Web application running, for example, inside a Google Chrome browser. Instead of separating tasks into threads, Google Chrome activates separate processes for all tasks. Let us assume that the Web application has two processes: user login (authentication/authorization) and search. Furthermore, there is an in-memory cache on the server side with two memory blocks. Cache access time is constant and synchronized and both caches can be accessed simultaneously. The processes have queues and will generate new login and search requests as soon as earlier ones have been completed. Let us further assume (just for the purpose of lowering complexity) that there are no cache misses. A new request for the first cache module is generated with probability q_1 and for the second with probability $q_2 = 1 - q_1$. Let us model system state as the ordered pair (i, j), where i is the number of requests waiting for the first cache module and j the number of requests waiting for the second module. The DTMC corresponding to this scenario is given in Figure 3.9.

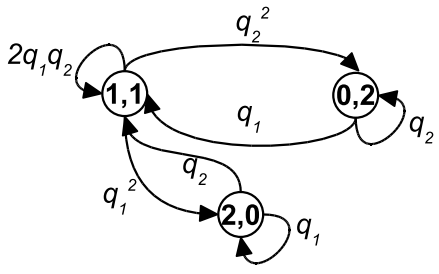

Figure 3.9: DTMC describing a multithreaded web application

In state (1,1) there is one request for each cache module and both can be served simultaneously. After that, both login and search processes generate

another request. The probability of both generating a request for the second cache block is q_2^2, and the chain transfers to state (0,2). The probability that both processes generate a request for the first cache block is q_1^2 and the chain transfers to state (2,0). If the processes generate requests for different cache modules, the chain remains in state (1,1) with probability $2q_1q_2$. In state (0,2) only one request can be served, therefore only one process can generate a new request. If this is again request for the second cache module, the chain remains in (0,2) with probability q_2, else it transfers to (1,1) with q_1. The analogous reasoning is applied to state (2,0).

Mapping the states (1,1), (2,0) and (0,2) into indices 0, 1 and 2, the following one-step transition probability matrix can be defined for this chain:

$$\mathbf{P} = \begin{bmatrix} 2q_1q_2 & q_1^2 & q_2^2 \\ q_2 & q_1 & 0 \\ q_1 & 0 & q_2 \end{bmatrix}$$

Based on the matrix, we can set up the Chapman-Kolmogorov system of equations:

$$\begin{aligned} \pi_0 &= \pi_0 2q_1q_2 + \pi_1 q_2 + \pi_2 q_1 \\ \pi_1 &= \pi_0 q_1^2 + \pi_1 q_1 \\ \pi_2 &= \pi_0 q_2^2 + \pi_2 q_2 \\ 1 &= \pi_0 + \pi_1 + \pi_2 \end{aligned}$$

Solving this system we obtain the limiting probability vector:

$$\begin{aligned} \pi_0 &= \frac{q_1q_2}{q_1q_2 + q_1^3 + q_2^3} \\ \pi_1 &= \frac{q_1^3}{q_1q_2 + q_1^3 + q_2^3} \\ \pi_2 &= \frac{q_2^3}{q_1q_2 + q_1^3 + q_2^3} \end{aligned}$$

This example demonstrates how to construct a DTMC and to calculate state probabilities. However, we still did not assign reliability and availability semantics to probabilities. We will do this in the following section.

Analysis of Continuous Time Markov Chains

Let us observe a repairable system with the failure rate λ and the repair rate μ. We can model such system using the 2-state CTMC shown in Figure 3.10.

The state 1 describes a functioning system (up state), while the state 0 is a failed system (down state). The generator matrix $\mathbf{Q}$ is given as:

$$\mathbf{Q} = \begin{bmatrix} -\mu & \mu \\ \lambda & -\lambda \end{bmatrix}$$

The Kolmogorov differential equation system is:

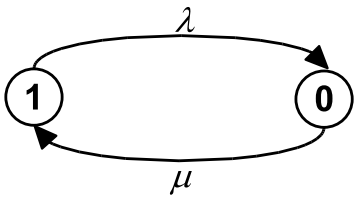

Figure 3.10: CTMC describing a repairable system

$$\pi_0'(t) = -\mu\pi_0(t) + \lambda\pi_1(t)$$

$$\pi_1'(t) = \mu\pi_0(t) - \lambda\pi_1(t)$$

We also know that $\pi_0(t) + \pi_1(t) = 1$, therefore the system can be rewritten:

$$\pi_1'(t) + (\mu + \lambda)\pi_1(t) = \mu$$

$\pi_1(t)$ is of interest, as it represents transient probability function that the system is operational, that is, instantaneous availability function. The equation above is linear differential equation of order one which has a standard method of analysis and solution (see e.g., [79]). After multiplying both sides with the integrating factor $e^{(\mu+\lambda)t}$, applying the property of product derivation and using the initial condition $\pi_1(0) = 1$ (the component is in the operational state at time 0), we obtain instantaneous availability function:

$$\pi_1(t) = A(t) = \frac{\mu}{\mu + \lambda} + \frac{\lambda}{\mu + \lambda}(e^{-(\mu+\lambda)t})$$

Well-known steady state availability formula which we gave without proof in section 2.1, can be obtained by taking the limit of this function when $t \to \infty$:

$$\pi_1 = A_s = \lim_{t\to\infty} A(t) = \frac{\mu}{\mu + \lambda}$$

If $U(t)$ is the uptime of the system in the interval $(0, t)$, expected uptime can be now easily calculated:

$$E[U(t)] = \int_0^t \pi_1(x)dx = \frac{\mu t}{\mu + \lambda} + \frac{\lambda}{(\mu + \lambda)^2}(1 - e^{-(\mu+\lambda)t})$$

Finally, instantaneous availability is given as:

$$A_I(t) = \frac{E[U(t)]}{t} = \frac{\mu}{\mu + \lambda} + \frac{\lambda}{t(\mu + \lambda)^2}(1 - e^{-(\mu+\lambda)t})$$

In the following sections we discuss more complex Markov availability models.

Markov Analysis of a 2-Component Non-Repairable System

Let us observe a two component non-repairable system. Such a system can be modeled with Markov chain shown in Figure 3.11. In state *11* both components are functioning. In states *10* and *01* one component has failed. Finally, in state *00* both components have failed.

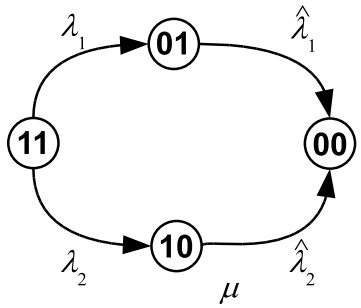

Figure 3.11: 2-Component non-repairable system

The generator matrix of this system is:

$$\mathbf{Q} = \begin{bmatrix} -(\lambda_1+\lambda_2) & \lambda_1 & \lambda_2 & 0 \\ 0 & -\hat{\lambda}_2 & 0 & \hat{\lambda}_2 \\ 0 & 0 & -\hat{\lambda}_1 & \hat{\lambda}_1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

States *11*, *01*, *10* and *00* are represented with indices *1*, *2*, *3* and *4* respectively. The corresponding Kolmogorov differential equation system is:

$$\begin{aligned} \pi_1'(t) &= -(\lambda_1+\lambda_2)\pi_1(t) \\ \pi_2'(t) &= \lambda_1\pi_1(t) - \hat{\lambda}_2\pi_2(t) \\ \pi_3'(t) &= \lambda_2\pi_1(t) - \hat{\lambda}_1\pi_3(t) \\ \pi_4'(t) &= \hat{\lambda}_2\pi_2(t) + \hat{\lambda}_1\pi_3(t) \end{aligned}$$

Solution of the system is:

$$\begin{aligned} \pi_1(t) &= e^{-(\lambda_1+\lambda_2)t} \\ \pi_2(t) &= \frac{\lambda_1}{\lambda_1+\lambda_2-\hat{\lambda}_2}(e^{-\hat{\lambda}_2 t} - e^{-(\lambda_1+\lambda_2)t}) \\ \pi_3(t) &= \frac{\lambda_2}{\lambda_1+\lambda_2-\hat{\lambda}_1}(e^{-\hat{\lambda}_1 t} - e^{-(\lambda_1+\lambda_2)t}) \\ \pi_4(t) &= 1 - (\pi_1(t) + \pi_2(t) + \pi_3(t)) \end{aligned}$$

These results can be used to calculate reliability of series configuration, where reliability is the probability that both systems function, that is, probability that the system is in state *1*:

$$R_S(t) = \pi_1(t) = e^{-(\lambda_1+\lambda_2)t}$$

Reliability of parallel configuration is the probability that both systems work or that either of systems work, that is, probability that the system is in states *1*, *2* or *3*:

$$R_P(t) = e^{-(\lambda_1+\lambda_2)t} + \frac{\lambda_1}{\lambda_1+\lambda_2-\hat{\lambda}_2}(e^{-\hat{\lambda}_2 t} - e^{-(\lambda_1+\lambda_2)t}) + \frac{\lambda_2}{\lambda_1+\lambda_2-\hat{\lambda}_1}(e^{-\hat{\lambda}_1 t} - e^{-(\lambda_1+\lambda_2)t})$$

If we assume that $\lambda_1 = \hat{\lambda}_1 = 1/3000$ and $\lambda_2 = \hat{\lambda}_2 = 1/2000$, solving the model for parallel configuration in Sharpe gives reliability graph given in Figure 3.12 with $MTTF = 3800$.

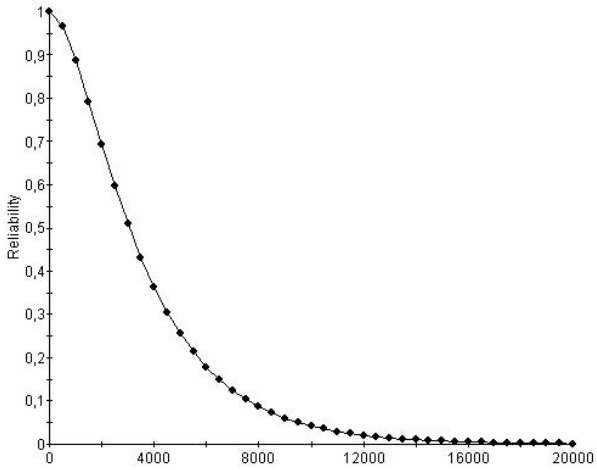

Figure 3.12: Reliability of 2-Component parallel non-repairable system

Markov Analysis of a Multicomponent System with Shared Repair

Let us observe a 2-Component repairable system where both components have identical failure rates λ. If one component fails, it can be repaired with repair rate μ. If, during repair, the second component also fails, the system has failed. In this case, we will analyse only reliability, that is, we do not consider the case where both components have failed, but can be repaired. Markov model representing this scenario is given in Figure 3.13.

The generator matrix of this system is:

$$\mathbf{Q} = \begin{bmatrix} -2\lambda & 2\lambda & 0 \\ \mu & -(\lambda+\mu) & \lambda \\ 0 & 0 & 0 \end{bmatrix}$$

Setting up the corresponding Kolmogorov differential equation system we get:

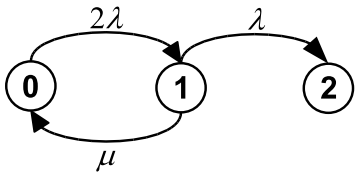

Figure 3.13: 2-Component repairable system

$$\pi_0(t) \ = -2\lambda\pi_0(t) + \mu\pi_1(t)$$
$$\pi_1(t)' = 2\lambda\pi_0(t) - (\lambda+\mu)\pi_1(t)$$
$$\pi_2(t) = \lambda\pi_1(t)$$

Solving the system enables us to calculate reliability, that is, the probability that system is not in state 2:

$$R(t) = 1 - \pi_2(t) = \frac{\alpha_2}{\alpha_1 - \alpha_2}e^{-\alpha_1 t} - \frac{\alpha_1}{\alpha_1 - \alpha_2}e^{-\alpha_2 t}$$

where α_1 and α_2 are the roots of:

$$s^2 + (3\lambda + \mu)s + 2\lambda^2 = (s + \alpha_1)(s + \alpha_2)$$

If $\lambda = 0.0001$ and $\mu = 0.001$, the mean time to failure is $MTTF = 65000$.

Let us now consider analogous 2-component availability model. We will assume that even if both systems failed, they can be repaired. We can distinguish between two repair strategies, shared and non-shared (both are shown in Figure 3.14). With shared repair, there is only one repair facility with repair rate μ which both systems have to share, and in the non-shared case there are two repair facilities with repair rate μ.

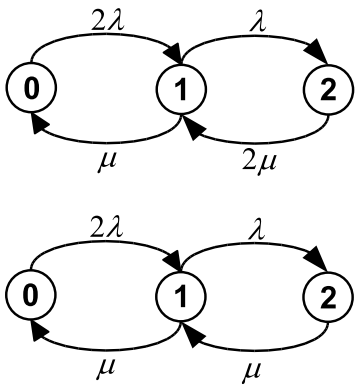

Figure 3.14: 2-Component Markov availability model with non-shared (above) and shared (below) repair

A non-shared case can be modeled with combinatorial methods (e.g., reliability block diagrams or fault trees), but shared case requires application of Markov chains. We will use steady-state balance equations, as we do not want to analyse transient availability in this example:

$$2\lambda\pi_0 = \mu\pi_1$$

$$(\lambda + \mu)\pi_1 = 2\lambda\pi_0 + \mu\pi_2$$

$$\mu\pi_2 = \lambda\pi_1$$

Solving this system for π_2, taking into account that $\pi_0 + \pi_1 + \pi_2 = 1$, gives:

$$\pi_2 = \frac{1}{1 + \frac{\mu}{\lambda} + \frac{\mu^2}{2\lambda^2}}$$

Steady state availability is the probability that a system is not in state 2, therefore:

$$A_S = 1 - \pi_2 = 1 - \frac{1}{1 + \frac{\mu}{\lambda} + \frac{\mu^2}{2\lambda^2}}$$

If we take $\lambda = 0.0001$ and $\mu = 0.001$, then $A_S = 0.9836065573770$ and $MTTF = 60000$. Transient availability graph is given in Figure 3.15.

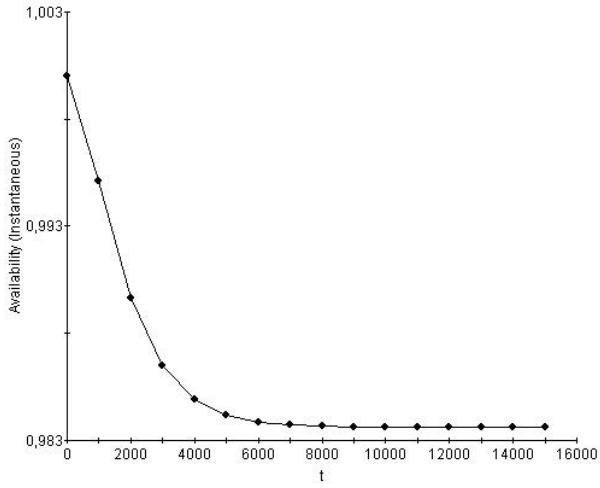

Figure 3.15: Availability of 2-component system with shared repair

Finally, to conclude investigation of repairable systems, let us construct a Markov model that enables comparison of different repair strategies. Assume that we have a 3-component system with identical failure rates λ and one repair facility with repair rate μ. If we want to improve system availability we could introduce additional repair facilities so that each system has its own repair facility with repair rate μ, or alternatively we could speed-up the single repair facility to 2μ. We want to answer the question which repair strategy is better

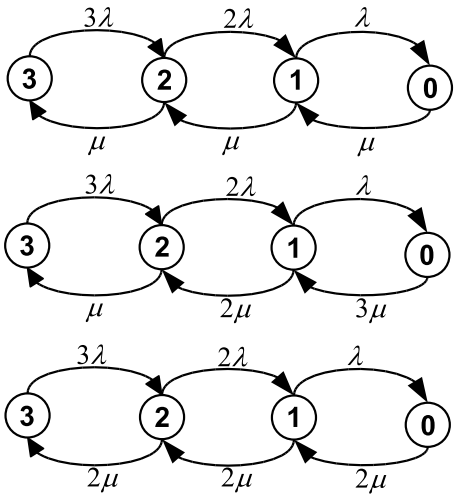

Figure 3.16: Markov model for comparing repair strategies

with respect to availability. To do so, we construct three Markov models shown in Figure 3.16.

Steady state analysis of these models yields steady state availability, mean time to failure, mean time to repair and downtime values given in Table 3.5.

	A_S	$MTTF$	$MTTR$	$Downtime$
Single repair (μ)	0.99561	226667	1000	2309
Non-shared repair (μ)	0.99925	443333	333	395
Single repair (2μ)	0.99936	776667	500	338

Table 3.5: Analysis results for comparing repair strategies

It is clearly visible that both strategies improve availability significantly (both add another nine to steady state availability). The speedup of the repair facility achieves slightly higher steady-state availability than introducing non-shared repaired facilities, but has also longer mean time to repair. Transient availability graphs for all three strategies are shown in Figure 3.17.

Markov models with imperfect coverage

Let us assume now that not all faults are recoverable. We introduce the coverage factor c, which represents conditional probability that the system recovers given that a fault has occurred. Let us observe a two-component parallel system with imperfect coverage, such as one proposed in [10] used to model an electronic switching system. Let us try to calculate system's mean time to failure. Markov model describing a system is shown in Figure 3.18. States 2, 1 and 0 represent two components being up, one component being up (failover successful with probability c), and system failure respectively. Failure and repair time are exponentially distributed with rates λ and μ respectively. If one component

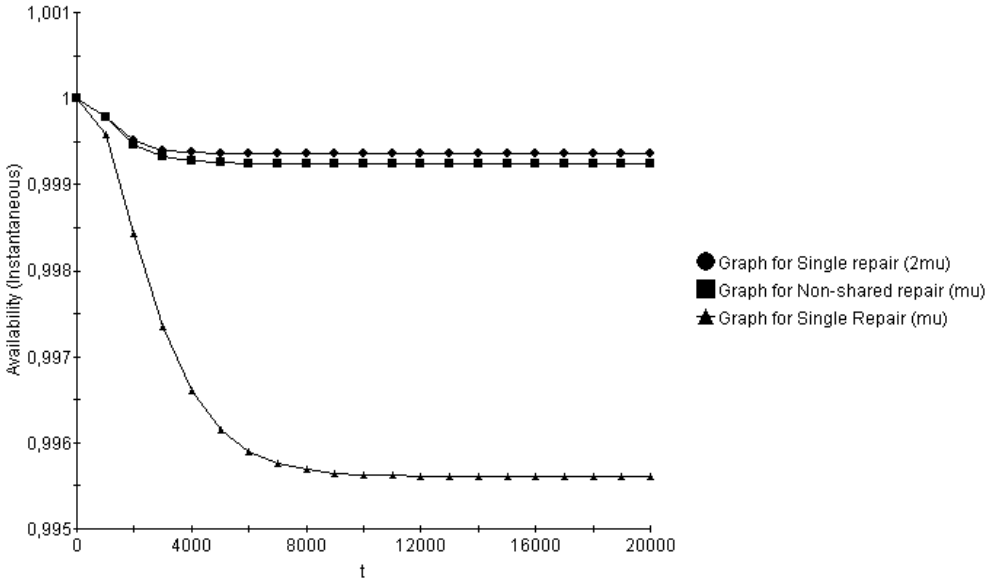

Figure 3.17: Transient availability for three repair strategies

fails, the system moves to state 1 with rate $2\lambda c$ instead of 2λ, as would be the case with perfect coverage (failover). The component that failed can be repaired and the system goes back to state 2 with probability μ, or the second component may fail with rate λ and the whole system fails (state 0). Of course, if the failover itself fails, the system goes from state 2 to state 0 directly with probability $2\lambda(1-c)$.

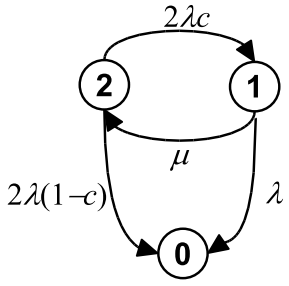

Figure 3.18: Two-component parallel system with imperfect coverage

The generator matrix of this system is:

$$\mathbf{Q} = \begin{bmatrix} -(2\lambda c + 2\lambda(1-c)) & 2\lambda c & 2\lambda(1-c) \\ \mu & -(\lambda+\mu) & \lambda \\ 0 & 0 & 0 \end{bmatrix}$$

The resulting Kolmogorov differential equation system is:

$$\pi_2(t)' = -2\lambda c\pi_2(t) - 2\lambda(1-c)\pi_2(t) + \mu\pi_1(t)$$

$$\pi_1(t)' = 2\lambda c\pi_2(t) - (\lambda + \mu)\pi_1(t)$$
$$\pi_0(t)' = 2\lambda(1-c)\pi_2(t) + \lambda\pi_1(t)$$

Assuming that the system is initially in state 2 (that is, $\pi_2(0) = 1$, $\pi_1(0) = \pi_0(0) = 0$), the system can be solved and reliability calculated as:

$$R(t) = \pi_1(t) + \pi_2(t)$$

Mean time to failure can be obtained now as:

$$MTTF = \int_0^\infty R(t)dt = \int_0^\infty \pi_1(t) + \pi_2(t) = \frac{\lambda(1+2c) + \mu}{2\lambda(\lambda + \mu(1-c))}$$

Figure 3.19 shows the reliability function for different coverage values c (λ=1/3000 and μ=1/2).

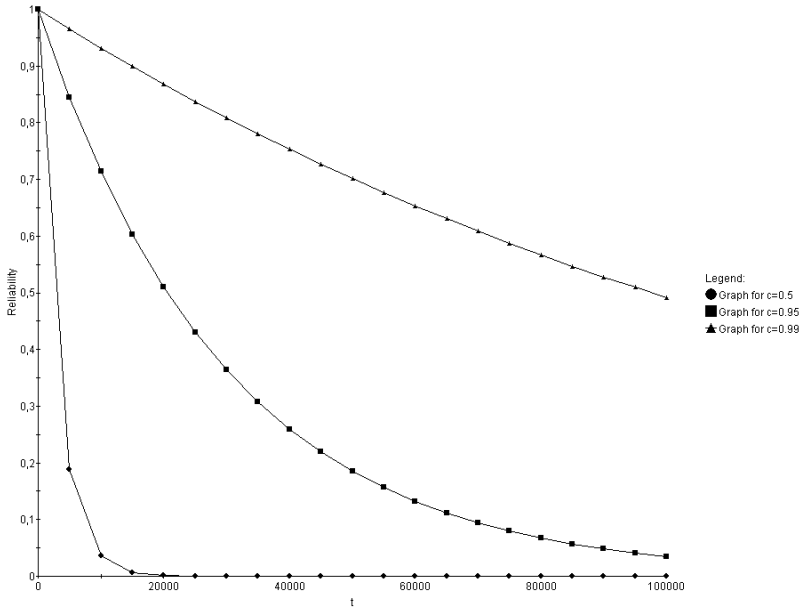

Figure 3.19: Reliability function for different coverage values

3.1.5 Stochastic Petri Nets

Markov models, presented in previous section, have several inherent limitations. As we already saw, and as is additionally demonstrated in Section 5.6, the state space grows much faster than the number of components in the system being evaluated. Also, Markov model is sometimes far removed in shape and feel from the physical system it is supposed to represent. One alternative is to use Petri net models, which offer a more concise formalism, closer to human intuition. Unfortunately, the solution complexity is thus not reduced.

We will first give an informal (descriptive) definition of Petri nets, followed by the more formal one, including various model extensions. A Petri net consists

of places, transitions, arcs and tokens. Tokens reside in places and move between them along the arcs and through the transitions. A marking is the number of tokens in each place. In pure Petri nets, transitions are untimed, and the measure of interest is the sequence of transition firings (releasing of a token). A very common extension are timed Petri nets, such as stochastic Petri nets (SPN)[149][140], where transitions are timed, meaning that the token firing is described as a stochastic process and characterized by appropriate distribution. If both immediate and timed transitions are allowed within one model, it is called a generalized stochastic Petri net (GSPN)[141].

Figure 3.20 shows a simple SPN model of an $M/M/1/k$ queue, where up to k (in this case $k = 5$) requests can be stored in a single queue. Requests arrive with constant rate λ and are served by a single server with rate μ (both arrival and serve times are exponentially distributed).

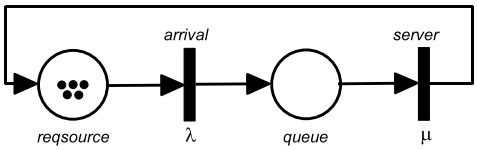

Figure 3.20: SPN model of an M/M/1/5 queue

The model has two places *reqsource* and *queue* modeling source of requests with five tokens and a queue with no tokens. Furthermore, two timed transitions *arrival* and *server* exist. After *arrival* has been enabled, which happens after random time sampled from the exponential distribution with parameter λ, it fires and a token moves from *reqsource* to *queue*. Firing time of *server* is exponentially distributed with parameter μ and after it fires, the token representing processed request is moved back to *reqsource*.

This SPN has six possible markings. If we represent each marking as an ordered pair (i, j) where i is the number of tokens in *reqsource* and j is the number of tokens in *queue*, the markings are (5,0), (4,1), (3,2), (2,3), (1,4) and (0,5). The set of all possible markings of a Petri net is called a reachability set. If we connect two markings by arcs if one can result from the firing of some transition enabled in the other, we obtain a reachability graph, which is unique for each initial marking of a Petri net. The reachability graph for the Petri net from Figure 3.20 is given in Figure 3.21.

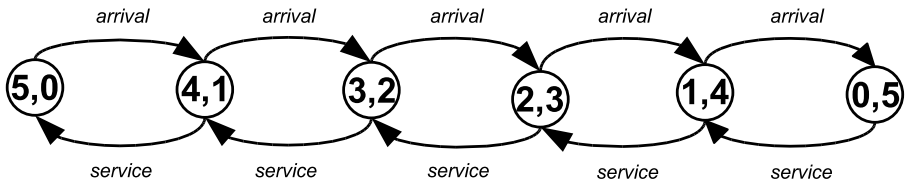

Figure 3.21: Reachability graph of the SPN model of an M/M/1/5 queue

Reachability graph is similar to Markov model. In fact, any SPN can be transformed into a stochastic process $M(t)$, where M is the net's marking, and analyzed using continuous-time Markov chain analysis methods. This is a common analysis method. It is worthwhile to note that SPN model is much more compact than the equivalent Markov model. If we want to extend the number of jobs, we would add new states to the Markov chain, but for the Petri net we just write new token number. While increasing number of jobs does not increase the Petri net size, it increases its corresponding reachability graph, which means that the model solution is thus not simplified. Very small Petri nets can generate enormous reachability graphs and it does not follow that the concise graphical modeling notation automatically implies faster model solution.

Let us give a formal Petri net definition now. A Petri net is a 5-tuple $(P, T, I(.), O(.), m_0)$ where P is a set of places, T is a set of transitions, $I(.)$ is the input function which maps transitions to multisets of places, $O(.)$ is the output function which maps transitions to multisets of places, and m_0 is the initial marking. A transition t is enabled by a marking m if and only if $I(t)$ is a submultiset of m. An enabled transition t may fire. Upon firing, $I(t)$ is subtracted from m and $O(t)$ is added to m, resulting in new marking. If a marking enables more than one transition, any of them may fire. The firing may then disable some transitions which were previously enabled.

Originally, Petri nets did not have a time element. If a net is not timed, the analysis is concerned with enabled markings, investigating firing order and examining if there are markings with special properties, e.g., absorbing markings in which no transition is further enabled. It is obvious that for dependability analysis, it is critical to introduce time to Petri nets. It can be done in either of two ways: time can be associated with places or with transitions. A SPN is a timed Petri net where time is associated with transitions. In a SPN, the transition is enabled as soon as all necessary tokens are assembled in required places. However, the transition does not fire right away as in non-timed Petri nets, but after a transition firing time which is specified by a distribution function. This time is measured from the instant the transition is enabled to the instant when it actually fires. In an SPN, it is assumed that the firing time is exponentially distributed. Thus each transition has a firing rate associated with it. The execution policy of a timed Petri net determines how the conflicting transitions are processed. The SPN uses race policy, where transition whose firing time elapses first is the one that fires first. An alternative is preselection policy. As already discussed, if both timed and immediate transitions are allowed in a net, it is a GSPN. When both timed and immediate transitions are enabled in a GSPN, only the immediate transitions may fire and the timed transitions behave as if they were not enabled. If more than one immediate transition is enabled, preselection is used (immediate transitions are assigned relative firing probabilities).

Petri Net Extensions

The basic Petri net model has been extended with many propositions over the years, however several extensions are today generally considered to be part of

the standard definition: arc multiplicity, inhibitor arcs, transition priorities, guards and marking-dependent arc multiplicity.

Arc multiplicity represents the case when more than one token is to be moved to or from a place. When an input arc connecting place p with transition t has multiplicity n, that means that t is enabled if and only if there are at least n tokens in p. Similarly, if an output arc connecting transition t to the place p has multiplicity n, then n tokens are moved to p when t fires. Graphical notation for arc multiplicity is the number placed next to the arc.

Inhibitor arc from place p to transition t disables t in any marking where p is not empty. Inhibitor arc can have multiplicity n in which case t is disabled whenever p has at least n tokens. Graphically, inhibitor arcs are represented as arcs with small circle instead of an arrowhead.

Priorities can be assigned to each transition as integer numbers. If more than one transition is possible, one with the lowest number, indicating higher priority, will be enabled. Priorities can be simulated using inhibitor arcs, leading to more complex models.

Guards can be added as a more general prioritizing mechanism, where each transition may have an additional enabling criterion. Transition is enabled only if the guard is satisfied. The guard can take any expression form, e.g., can impose a condition that the number of tokens in some places (or their sum) must be positive. Guards are usually written as textual expressions on the side of transitions. However, if there are many guards, or guards are complex, they may be maintained in a separate table.

Marking-dependent arc multiplicity enables to express arc multiplicity in a dynamical way, using expressions similar to guards. Essentially, number of tokens transferred can be dependent upon the system state. For example, if a token appears in a certain place, arc multiplicity may be configured based on that particular marking.

Finally, **marking-dependent firing rates** can be introduced, which allow transition firing rate (for GSPN) to depend on the particular Petri net marking. In the most general form, firing rate of any transition may depend on the number of tokens in any of the places. A more limited form is sometimes used, where constraint is allowed only on the number of tokens in one of the places with an outgoing arc leading to the transition. Usually, firing rate is a multiple of number of tokens (e.g., if a place contains n tokens representing operating components, than the transition modeling component failure will have the rate $n\lambda$).

GSPN Availability Model

As an example of GSPN availability modeling let us assume that we have n Web services working in a replicated configuration, where $k < n$ must be operational for the system to be considered operational. Furthermore, let us assume that there is only one repair facility for all services. The GSPN model representing this replicated service system is shown in Figure 3.22, assuming that service failure and repair rates are λ and μ respectively.

There are initially n tokens in place *up*. Transition $fault$ is timed and

depends on the number of tokens in place *up* (number of operating services), its firing rate is therefore denoted with $\lambda\#$. When a service fails, a token is moved to *down*. It can be repaired and moved back to *up* via transition *repair* which has rate μ. If $n-k+1$ tokens have accumulated in *down*, that is, less than k services are operating, immediate transition *crash* is activated which takes all remaining $k-1$ tokens from *up* as well as $n-k+1$ from *down* and moves them into place $failure$.

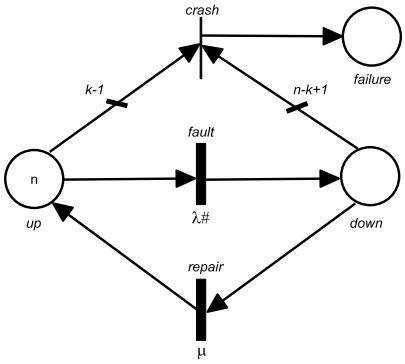

Figure 3.22: GSPN availability model of the k-out-of-n Web service system

The most commonly used method for analyzing stochastic and generalized stochastic Petri nets is generation and analysis of the stochastic process associated with the Petri net. The other approach is to perform discrete event simulation, which is useful when Petri net generates a large reachability graph. Let $M(t)$ be the marking of a Petri net at time t. $M(t)$ is a continuous stochastic process called the marking process or stochastic process underlying the stochastic Petri net. The state space of this process is the reachability set of the Petri net. If all transitions are timed and exponentially distributed, the marking process is a continuous time Markov chain, where the transition rate from marking m_j to m_k is the sum of rates of all transitions that are enabled in the former and whose firing generate the later marking. Hence, steady-state or transient analysis of such Petri net is equivalent to the solution of a CTMC, given in section 3.1.4.

Analysis of GSPN is more complex, caused by the presence of immediate transitions. A marking in which at least one immediate transition is enabled is called a vanishing marking. Otherwise, it is called a tangible marking. When the reachability graph of a GSPN is constructed, arcs corresponding to timed transitions are labeled with rates and arcs corresponding to immediate transitions are labeled with probabilities. The resulting graph is called an extended reachability graph (ERG). It is neither a CTMC nor a DTMC. However, after eliminating vanishing markings, and including their effects into transition rates between tangible markings, a CTMC is obtained.

Results of the numerical transient analysis of 2-out-of-3 Web service system (using SHARPE) are given in Figure 3.23. The equivalent reachability graph (Markov chain) is given in Figure 3.24. Note that if we choose to analyze 8-out-

of-10 system, Markov model would grow, whereas Petri net model remains the same in size. Furthermore, Petri net model enables easy simulation or varying of parameters, such as k and n in this example.

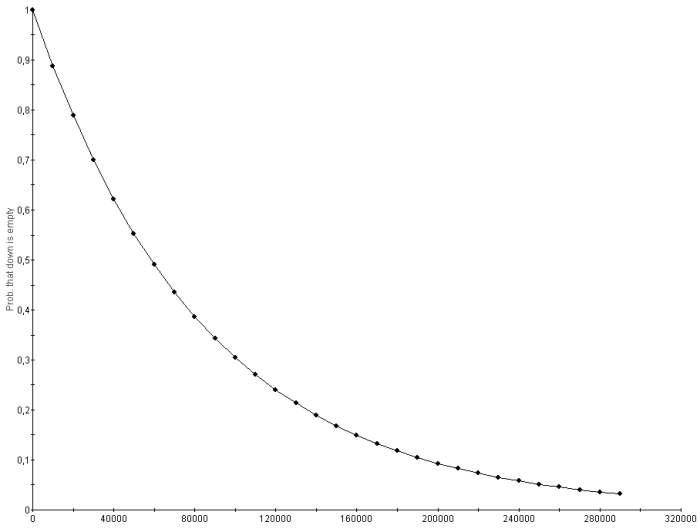

Figure 3.23: Transient availability analysis of 2-out-of-3 system

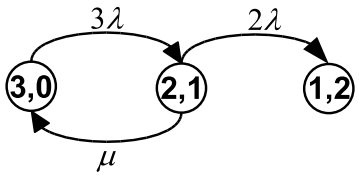

Figure 3.24: Reachability graph of 2-out-of-3 Petri net model

3.1.6 Stochastic Activity Networks

Stochastic Activity Networks (SAN) are a variant of stochastic Petri nets, based on non-probabilistic activity networks, similar to the way that untimed Petri nets provide foundation for stochastic Petri nets. In this section, we will first introduce activity networks, extend their definition to probabilistic models (SAN) and give examples of SAN reliability/availability models, solved using Möbius tool (more information available in Section B.1.24).

Activity Networks

Activity networks are generalized Petri nets with following primitives:

- Activities, which can be timed and instantaneous, and each of which has a non-zero integral number of cases. The term case is used to denote a possible action taken upon the completion of an event.

- Places, equivalent to Petri net places.
- Input gates, each of which has a finite set of inputs and one output. Each input has associated n-ary predicate and n-ary computable partial function over the set of natural numbers. They are called enabling predicate and the input function, respectively. The input function is defined for all values for which the enabling predicate is true.
- Output gates, each of which has a finite set of outputs and one input. With each output gate an n-ary computable function on the set of natural numbers is associated, called the output function.

Let us define some important additional concepts of activity networks. Suppose P is the set of all places in the network. If S is a set of places such that $S \subseteq P$, a marking of S is a mapping $\mu : S \to \mathbb{N}$. An input gate can now be defined as a triple (G, e, f) where $G \subseteq P$ is the set of input places associated with the gate, $e : M_G \to \{0, 1\}$ is the enabling predicate of a gate and $f : M_G \to M_G$ is the input function of a gate. Similarly, an output gate is a pair (G, f) where $G \subseteq P$ is the set of output places associated with a gate, and $f : M_G \to M_G$ is the output function of a gate.

Following these premises, an activity network is formally defined as an eight-tuple $AN = (P, A, I, O, \gamma, \tau, \iota, o)$ where P is a finite set of places, A is a finite set of activities, I is a finite set of input gates, O is a finite set of output gates. Further, $\gamma : A \to \mathbb{N}^+$ specifies the number of cases for each activity, and $\tau : A \to \{Timed, Instantaneous\}$ specifies activity type. Structure of an activity network is specified with function $\iota : I \to A$, which maps input gates to activities and $o : O \to \{(a, c)|a \in A \wedge c \in \{1, 2, ..., \gamma(a)\}\}$, which maps output gates to cases of activities.

The behavior of an activity network is a characterization of possible completions of activities, selection of cases and changes in markings. Input gate $g = (G, e, f)$ holds in a marking μ if $e(\mu_G) = 1$. Activity a is enabled in marking μ if g holds for all $g \in \iota^{-1}(a)$. Finally, marking μ is stable if no instantaneous activities are enabled in μ. The set of reachable markings of network AN in marking μ_0 is the set of markings $R(AN, \mu_0)$ where $R(AN, \mu_0) = \{\mu|\mu_0 \xrightarrow{*} \mu\}$. The set of stable reachable markings is the set $SR(AN, \mu_0) \subseteq R(AN, \mu_0)$ of reachable markings that are stable. An activity network is stabilizing in marking μ_0 if for every $\mu \in SR(AN, \mu_0)$ the set $S(\mu)$ is finite.

Based on these premises we now define stochastic extension of activity networks.

Definition of a Stochastic Activity Network

Given an activity network which is stabilizing in a given initial marking, a stochastic activity network (SAN) is formed by introducing additional functions C, F and G. Function C represents the probability distribution of case selections, F the probability distribution of activity delay times and G the set of reactivation markings for each possible marking. Let us first formally define a SAN and then discuss its definition using an example.

A stochastic activity network is a five-tuple $SAN = (AN, \mu_0, C, F, G)$. $AN = (P, A, I, O, \gamma, \tau, \iota, o)$ is an activity network. μ_0 is the initial marking and a stable marking in which AN is stabilizing. C is the case distribution assignment, which assigns functions to activities such that for any activity a, $C_a : M_{IP(a) \cup OP(a)} \times \{1, ..., \gamma(a)\} \rightarrow [0, 1]$. Furthermore, for any activity a and marking $\mu \in M_{IP(a) \cup OP(a)}$ in which a is enabled, $C_a(\mu, \cdot)$ is a probability distribution called the case distribution of a in μ. F is the activity time distribution function assignment. It assigns a continuous function to timed activities such that for any timed activity a, $F_a : M_P \times \mathbb{R} \rightarrow [0, 1]$. Furthermore, for any stable marking $\mu \in M_P$ and timed activity a that is enabled in μ, $F_a(\mu, \cdot)$ is a continuous probability distribution function called the activity time distribution function of a in μ. G is the reactivation function assignment. It assigns functions to timed activities such that for any timed activity a, $G_a : M_P \rightarrow \wp(M_P)$, where $\wp(M_P)$ denotes the power set of M_P. Finally, for any stable marking $\mu \in M_P$ and timed activity a that is enabled in μ, $G_a(\mu, \cdot)$ is a set of markings called the reactivation markings of a in μ.

Let μ, a, c be the marking, activity and case of a given SAN respectively. Execution of a SAN is a sequence of configurations $< \mu, a, c >$, where for each configuration, the SAN was in marking μ, activity a was completed and case c was chosen. In any marking μ the activity that completes is selected from the set of active activities in μ, that is, those activities that have been activated but not yet completed or aborted. After each activity completion and case selection, set of activities that are active is altered. If the marking reached is stable, then the activity that completed, activities that are no longer enabled and those for which the reached marking is a reactivation marking are removed from the set of active activities. Activities that are now enabled, but are not in the set of active activities, are placed in it (including those that are reactivated). If the marking reached is not stable, then timed activities are not added or deleted from the set of active activities. Instead, the activity that completed is removed, instantaneous activities that are no longer enabled are also removed, and instantaneous activities that became enabled but not in the set are added. The choice of the activity to complete from the set of active activities is determined by the activity time distribution function, and the relative priority of timed and instantaneous activities. If more than one instantaneous activities may complete, one is chosen non-deterministically. If there are none, then the timed activity with the earliest completion time is selected. The activity time distribution function defines the time between activation and completion. After the activity has been selected, a case is chosen based on the case distribution in the current marking. Note that non-deterministic choice of non-timed activities may yield non-probabilistic behavior of the network. Therefore, rules for well-specified and well-behaved networks are introduced [176]. We will not formally discuss these rules, but will implicitly further discuss only probabilistic networks.

Let us briefly compare graphical model elements of stochastic Petri nets and stochastic activity networks using a simple example. Suppose we want to model a priority queue, with two job classes, one of which has higher priority. Figure 3.25 shows both SPN and SAN model of such a system.

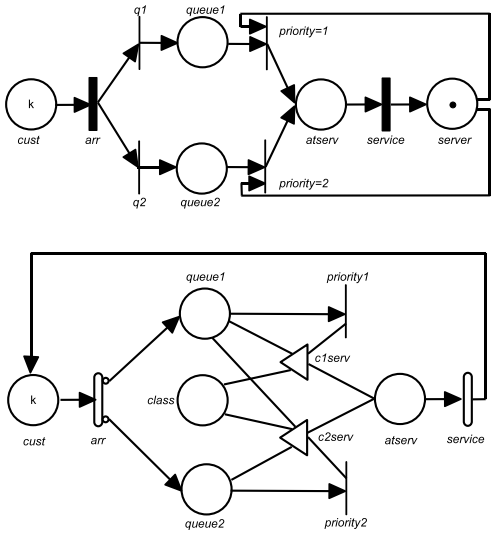

Figure 3.25: SPN and SAN models of a priority queue

In the SPN model, jobs of class i arrive with probability q_i and go to place $queue_i$. If all queues are full, jobs with higher priority (denoted with lower integer), are served first, as long as there is a token in the place *server*, because only immediate transition with highest priority will fire. Similarly, in the SAN model jobs arrive and are probabilistically assigned a class (two cases of activity *arr*), after which they are moved to $queue_i$. Instantaneous activity $priority_i$ is enabled if place $queue_i$ is marked and the predicate of $c_i serv$ is true. For example, the predicate for $c_2 serv$ is `(atserv->Mark()==0) && (queue1->Mark()==0)`. In other words, jobs of lower priority may be served only if the server is available (no tokens in *atserv* denoted with `atserv->Mark() ==0`) and there are no jobs with higher priority in $queue1$ denoted with `queue1-> Mark()==0`. Place $Class$ hold a value of either 1 or 2, representing the class of a job currently in service. This enables activity *service* to have a completion rate dependent of the job type being served. For example, firing of activity $priority2$ is described with: `atserv->Mark()++; class->Mark()=2; queue2->Mark()--;`. Also note that SANs use C++ syntax for describing markings.

SAN Availability Model

As an example of the SAN availability model, we will use the adapted version of a highly redundant fault-tolerant multiprocessor model first described in [126] and further elaborated in [174]. The model will be solved using Möbius tool, and all models shown here are directly exported from it.

The system under study consists of computers, each with three memory modules (one is a spare), three CPU units (one is a spare) two I/O ports (one spare) and two non-redundant error-handling chips. Each memory module consists of 41 RAM chips (two spares) and two interface chips. Each CPU and I/O port consists of six non-redundant chips. The overall system is considered

operational if at least one computer is operational; furthermore, a computer is operational if at least two memory modules, two CPU units, one I/O port and both error-handling chips are operational. The failover is not perfect, with following coverage probabilities: RAM chip 0.998, memory module 0.95, CPU unit 0.995, I/O port 0.99 and computer 0.95. The failure rate λ for each chip in the system is assumed to be 100 failures per billion hours [126].

We first describe the CPU submodel (Figure 3.26). The number of tokens in places *cpus* and $computer\_failed$ represent the number of operational CPUs and failed computers in the system, respectively. Place *cpus* is initialized with three and $computer\_failed$ with zero tokens. Place *ioports* represents the number of operational I/O ports and is therefore initialized to two tokens. Place *errorhandlers*, representing the number of functional error-handling chips is initialized to two.

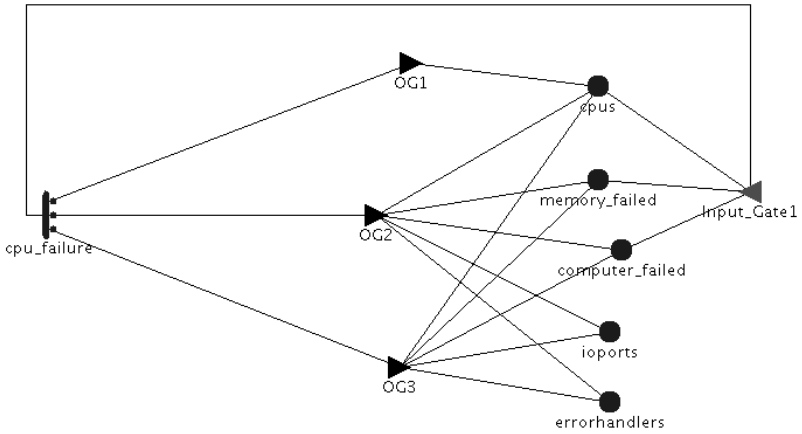

Figure 3.26: SAN model of the CPU

The CPU failure is modeled with timed activity $cpu\_failure$. The activity completion rate is defined as `6.0 * failure_rate * cpus->Mark()`, that is, the activity completion rate is equal to the six times the failure rate of a single chip (`failure_rate`) times the number of currently operational processors, which is the current marking of place *cpus* (`cpus->Mark()`). If a spare CPU is available, three cases are associated with the completion of this activity. The first case is a successful coverage of the CPU failure. In this case, failed CPU is replaced by a spare, and the computer continues operation. The second case represents a CPU failure that is not covered, but the computer failure caused by it is covered. If this happens and a spare computer is available, failed computer is replaced by a spare and the system continues to operate. However, if no computer spare is available, the systems fails. Finally, the third case represents situation where both CPU and computer failure are not covered and the overall system fails as a consequence. If no spare CPU is available, then a CPU unit always causes a computer failure (this situation should not be mixed with three previous cases, where a spare was always available but with imperfect coverage factor). In this case, two possible outcomes may happen: if a spare computer

is available, the fault can be covered. If no spare is available, the system fails. In table 3.6, which specifies all mentioned situations, these cases are guarded by the statement `if (cpus->Mark()==3)`.

Case	Probability

1	`if (cpus->Mark()==3)`
	` return(CPU_cov);`
	`else`
	` return (0.0);`

2	`if (cpus->Mark()==3)`
	` return((1.0-CPU_cov)*comp_cov));`
	`else`
	` return(comp_cov);`

3	`if (cpus->Mark()==3)`
	` return(1.0-CPU_cov)*(1.0-comp_cov));`
	`else`
	` return (1.0-comp_cov);`

Table 3.6: Case probabilities for $cpu\_failure$ activity

The input gate ($Input\_Gate1$) is used to determine if timed activity $cpu\_failure$ is enabled in the current marking and can complete. It can be enabled if at least two working CPUs are available and no more than two memory modules have failed, as well as if the overall system has not failed. This enabling predicate is formally represented as `cpus->Mark()>1)&&(memory_failed->Mark()<2) && (computer_failed->Mark()< num_comp)`.

Finally, output gates $OG1$, $OG2$ and $OG3$ are used to determine the next marking based on the current marking and the case chosen when $cpu\_failure$ completes. $OG1$ covers case 1, where CPU failure is successfully covered. In this case, the number of operating CPU units is decremented. $OG2$ covers case 2, where CPU failure is not covered, but the computer failure is covered. Actions required in this case are to set the number of processors, I/O ports and error-handling chips to zero, number of failed memory units to two (this is the definition of a failed computer) and to increment the number of failed computers. $OG3$ represents the system failure event, where same actions as in $OG2$ are performed, with the exception that the number of failed computers is set to total number of computers (definition of a total system failure). C++ rules for the output gates are given in Table 3.7.

Output gate	Function
OG1	`if (cpus->Mark()==3)` ` cpus->Mark()--;`

OG2	`cpus->Mark()=0;`
	`ioports->Mark()=0;`
	`errorhandlers->Mark()=0;`
	`memory_failed->Mark()=2;`
	`computer_failed->Mark()++;`

OG3	`cpus->Mark()=0;`
	`ioports->Mark()=0;`
	`errorhandlers->Mark()=0;`
	`memory_failed->Mark()=2;`
	`computer_failed->Mark()=num_comp;`

Table 3.7: Output gate functions

We give the remaining atomic models (I/O port, error-handler and mem-

ory module) without further discussion in Appendix A and concentrate on the model solution here. Previously defined atomic models must, however, first be composed into a system model. Note that this is not a hierarchical composition, such as the composition of combinatorial and non-combinatorial models (Section 5.7 gives an example of the hierarchical composition of fault trees and Markov chains), but plain composition of models at the same abstraction level specified using the same formalism. SANs offer two primitives for model composition: replicate and join. The former creates a given number of atomic models, while the latter joins state space and structure of more models into a single representation. The composite model for this scenario is given in Figure 3.27.

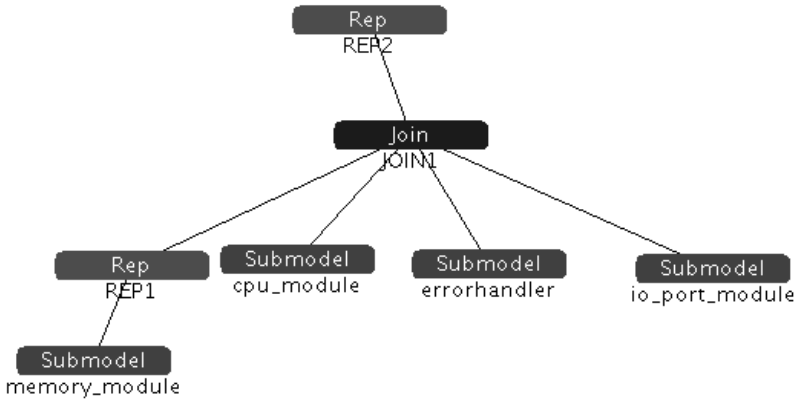

Figure 3.27: Composed SAN model

To solve the model, a reward variable has to be defined next. We will focus on reliability and determine system reliability for the variable mission time parameter. The system is considered unreliable at time t if all *num_comp* computers have failed by that time, that is, if there are *num_comp* tokens in place $computer\_failed$. We define reward variable *unreliability* by specifying its reward function for all submodels of the system (in this example, for all computers):

```
if (cpu_module->computer_failed->Mark()) == num_comp)
{
    return 1.0/num_comp;
}
```

Basically, reward of `1/num_comp` is accumulated once for each computer, or a total of *num_comp* times. This gives the total reward of 1. Hence, system unreliability can be calculated as the mean of this reward variable.

SAN model can be solved analytically or using simulation. Analytical solution can be obtained using any of the following existing methods: direct steady-state solver which uses a numerically stable version of LU decomposition [206]; iterative steady-state solver using successive over-relaxation and Jacobi methods; Takashi steady-state solver [189] which is an instance of general class

of iterative solvers called IAD (Iterative Aggregation/ Disaggregation) solvers; deterministic iterative steady-state solver [182] which uses uniformization and successive over-relaxation; advanced deterministic iterative steady-state solver [139] which uses two-stage iterative method that takes advantage of the structure of the probability transition matrix of an embedded Markov chain; transient solver which calculates the mean and variance of each performability variable for time points defined for the reward variable within the reward model; accumulated reward solver which gives the expected accumulated reward, as well as the expected time-averaged accumulated reward over the interval; adaptive transient solver which uses the same method as the transient solver, but is more efficient for stiff models in which there are large (orders of magnitude) differences in the rates of actions because it uses adaptive uniformization [210].

In this example we give results using the transient solver as well as discreet-event simulation for experiments containing 1, 2 and 3 computers for estimating reliability after 10 and 20 years of operation. Results are in Table 3.8.

time	number of computers					
	1		2		3	
	unreliability					
	trans	sim	trans	sim	trans	sim
10	0.1194	0.1263	0.1077	0.1073	0.1097*	0.1108
20	0.2079	0.2110	0.1218	0.1189	0.1102**	0.1161

Table 3.8: Solution of the SAN model using transient solver and simulation

Note that transient analysis for 3 computers (values marked with * and ** in the table), require up to six hours of overnight (low server load) state space generation on the Sun Fire server with two dual core Opteron CPUs running virtualized Linux. This represents a serious issue when using analytical solving of models which generate large state space.

3.1.7 Markov Reward Models

Complex SOA systems are sometimes gracefully degrading systems, that is, they are able to survive the failure of one or more components without immediately ceasing to operate. Instead they continue to provide a service, but at reduced quality level. This may or may not be the consequence of inherent fault-tolerance: for example, a system may still be working simply because of the existing (unforseen) redundancy, or because the service in question does not use the part of infrastructure which has failed. Such systems cannot be completely modeled using combinatorial and state-space approaches presented so far. Instead, performability models are used to describe behavior of gracefully degrading systems. One of the most commonly used models for this purpose is the Markov reward model (MRM).

Let us motivate the need to discuss performability with a simple and intuitive example. Assume that we have a system of n processors placed in parallel and m buffers connected in series. Obviously, reliability and availability of such a system will increase as the number m is decreased. This is intuitively clear,

as adding more serial components increases the probability that one of them will fail. If the processor failure rate is λ and the buffer failure rate is γ, the analytical expression of the reliability function $R(t) = (1 - (1 - e^{-\lambda t})^n)e^{-m\gamma t}$ confirms the intuitive conclusion. If we use only this line of reasoning, we may conclude that the optimal system configuration is that with minimal number of buffers (minimal m). However, this will likely impact the throughput of the system, as the processors will be idle while the buffers are full. In order to increase the throughput, we would ideally like to incorporate as many buffers as possible. This will, on the other hand, reduce system availability. Based on this simple thought experiment it is clear that we need some way to capture both aspects of system behavior within one model.

If a component of a gracefully degradable system (further: degradable system) fails, the system will either reconfigure itself and continue operating in a degraded state of operation with reduced capacity/performance, or will completely fail. A degradable system may have several reduced states between fully operational and failed, and performance/capacity level is associated with each state. Some degradable systems may allow for failed components to be repaired on-line, while operating in a degraded state, thus returning the system into fully functional state once repair is complete. The performance level of a degradable system can therefore vary with time, both decreasing (failure) and increasing (repair), as sketched in Figure 3.28.

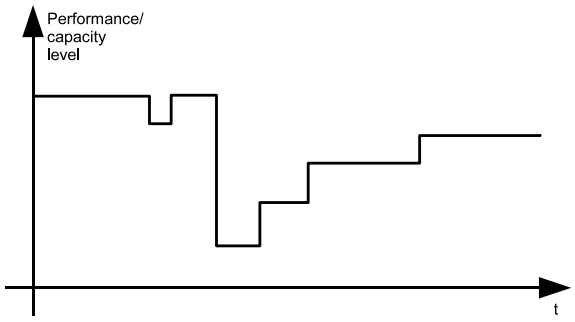

Figure 3.28: Performance/capacity level of a degradable system in time

The measures of interest, that we may ask from the model describing such a system, are: expected performance at time t including effects of failure, repair and resource contention; time-averaged performance of the system in the interval $(0, t)$; probability that n jobs have been serviced by time t. Markov reward model can answer these questions.

Definition of a Markov Reward Model

It is possible to construct a single Markov model which would incorporate both structural variations (failure/reconfiguration/repair) as well as performance level of the system under study (arrival/completion of jobs). However, such a model would reach a prohibitive size very quickly. Also, due to different orders

of magnitude between occurrence rates of performance- and reliability-related events, analytical solutions are problematic (the problem is known as stiffness of a model). It is therefore recommended to decompose performability model into two parts: one to capture structural variations, which is the structure state model, and one to capture performance, which is the reward model. Note that by dividing a system model into two models, we actually introduce an approximation, because obviously, the speed with which a job is being serviced (part of the reward model) will depend on the number of operating components providing a service (part of the structure state model). In other words, two models are not really independent. However, because of the mentioned order of magnitude difference between occurrence rates (e.g., performance rates in milliseconds or seconds, reliability in hours, days, years), the performance measure will likely reach a steady-state condition before any failure occurs. Decomposition into two models is therefore an accepted procedure.

Decomposition is performed hierarchically: structure-state availability model is constructed first and for each state a reward model is generated, which is performance model within the stationary structural state. The overall measure is then obtained by combining performance measures and state probabilities. Markov reward model enables us to characterize each state of the system with a reward index or rate, which denotes performance level in that state.

Formally, a Markov reward model comprises a continuous time Markov chain $X = \{X(t), t \geq 0\}$ with a finite state space S and a reward function r where $r : S \rightarrow \Re$. X is described by its generator matrix Q and initial probability vector $\pi(0)$ as defined in Section 3.1.4. For each state $i \in S$, r_i represents the reward obtained per unit time spent by X in state i. We will assume that the reward associated with a state describes performance level of a system in that state. The value of reward can be negative, it denotes a loss instead of gain then. This model is called a rate-based reward model because the system produces work at a rate r_i while in state i. There are also impulse-based reward models in which there is an impulse reward $r(i, j)$ associated with transition from i to j. In those models, reward is accumulated instantaneously on state transition. Both models (rate- and impulse-based) support time-dependent rewards. Some formalisms and tools allow even to combine both models (e.g., stochastic activity networks, described in the previous section).

A Markov reward model can calculate three types of steady-state and transient measures: expected value of reward rate, expected accumulated reward, and distribution of accumulated reward. The community still debates which of those measures is strictly performability per definition, but that discussion is outside of scope and we will briefly cover all three here.

The expected value of the reward rate is the average of all possible rate values in all possible states. The steady state measure is defined as follows. Let $\pi = [\pi_0, ..., \pi_i, ...\pi_n]$ be the vector of Markov chain state probabilities at steady state. Let $Z(t) = r_{X(t)}$ be the system reward rate at time t. The expected steady-state reward rate is then defined as:

$$E[Z] = \sum_{i \in S} r_i \pi_i$$

Similarly, let $\pi(t) = [\pi_0(t), ..., \pi_i(t), ..., \pi_n(t)]$ be the vector of state probabilities at time t. The expected instantaneous reward rate is defined as:

$$E[Z(t)] = \sum_{i \in S} r_i \pi_i(t)$$

Note that if we let r_i be 1 for all operational states and 0 otherwise, the equations given above transform into instantaneous and steady-state availability, which conforms to the intuitive discussion from previous section.

The accumulated reward in the interval $[0, t)$ is denoted by $Y(t)$ and is defined as:

$$Y(t) = \int_0^t Z(\tau) d\tau$$

If the system has perfect repair, that is, Markov process does not have absorbing states, $\lim_{t \to \infty} Y(t)$ cannot be defined since $Y(t)$ increases indefinitely. This is because a system with perfect repair can accomplish any finite task, given enough time of course. Therefore, another measure is often used, called averaged cumulative reward $W(t) = Y(t)/t$.

An example Markov reward model, its corresponding stochastic process $X(t)$, expected value of the reward rate $Z(t)$ and the expected accumulated reward $Y(t)$ are given in Figure 3.29.

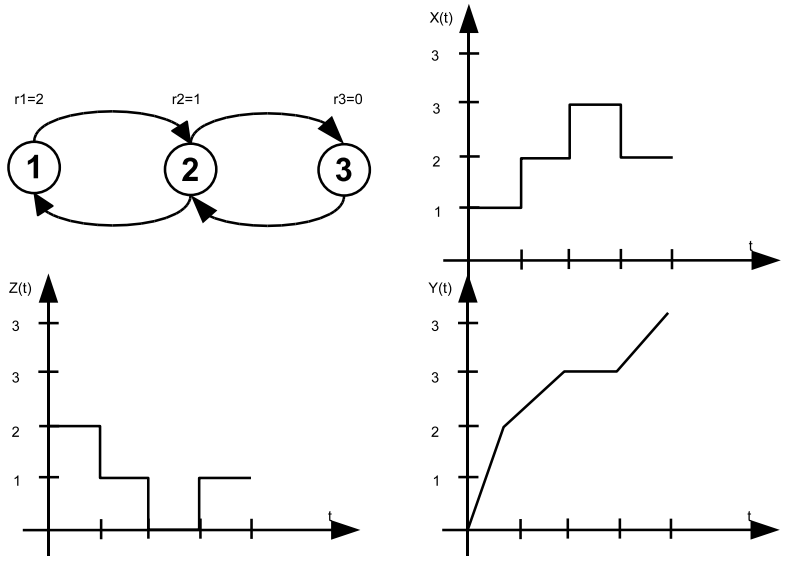

Figure 3.29: Markov reward model, expected and accumulated reward

Finally, the distribution function of accumulated reward is defined by:

$$F(t, y) = P\{Y(t) \leq y\}$$

The complementary distribution $F^c(t, y) = 1 - F(t, y) = P\{Y(t) > y\}$ enables us to calculate probability that the system is able to achieve a given

amount y of work during the interval $[0, t)$. If $t \to \infty$, distribution of the accumulated reward can be defined only for a system with imperfect repair using a well known method from [19]. The method is based on the transformation of Markov chain into an equivalent one with the same state space, but with new transition rate obtained by dividing the corresponding one of the original chain by the reward rate of departing state. It can be shown that the limiting distribution of $Y(t)$ can be obtained then from the transient state probabilities of the new chain. Computation of the distribution of accumulated reward over a finite time is much more difficult, and some solutions can be found in [68], [204].

The last issue to discuss before we show an example of Markov reward model is how to assign rewards to states. There is neither straightforward answer to this question nor universal models. There are some attempts to generate reward models automatically (such as [60] and [102]), but that discussion is out of scope. Instead we summarize some possible meanings of rewards in standard practice and then give a concrete service performability example. As a consequence however, we will not be able to derive performability models in Chapter 5 automatically. The simplest case of reward assignment is to make the reward rate a function of the state index. For example, if we assign a reward rate 1 to all operational states, and 0 otherwise, then $E[Z(t)]$ gives instantaneous availability, $E[Z]$ gives steady state availability and $E[W(t)]$ gives interval availability. This is a pure availability model. Assume further than we have a 2 processor system with two memory modules and that we built a Markov chain where states are described with ordered pairs (i, j), i is the number of requests for the first memory module and j is the number of requests for the second memory module. In other words in state (1,1) both requests can be served in parallel, and in states (2,0) and (0,2) only one memory request per cycle is possible. Let us assume that performance metric is the expected steady state memory bandwidth, then we define a reward as the number of requests processed in each state: $r_{11} = 2$ and $r_{20} = r_{02} = 1$. In other cases, where mix of reliability/availability and performance is required, other metrics such as the number of functioning units, throughput, response-time deadline violation probability, or buffer loss probability are usually used.

Markov Reward Model Example

Let us assume that we have a Web-based system with two Web services which receive user requests and three database servers which are then subsequently queried. Web services work in parallel and the first free service accepts the user request, and then uses one database server to produce an answer. The system is considered operational as long as there is at least one Web service and one database server operational. Web service and database server failure times are exponentially distributed with parameters λ_S and λ_D respectively. This system can be represented using Markov model given in Figure 3.30.

State (i, j) defines configuration with i functional Web services and j functional database servers. Let us discuss a possible reward model that can be associated with this Markov chain. The simplest case would be to assign re-

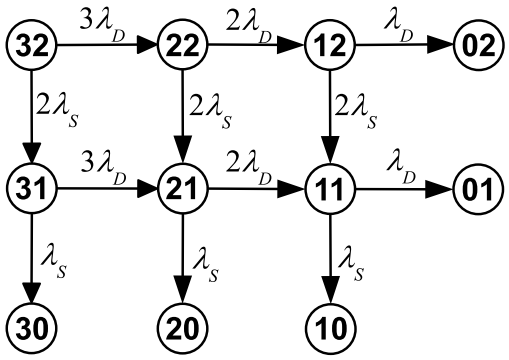

Figure 3.30: Markov reward model

ward one to all functional states (32, 22, 12, 31, 21, 11) and zero otherwise. However, we already showed that this results in simple reliability evaluation. A more accurate measure would be to describe some capacity or ability of the system to perform useful work. Let us assume further that database access is costlier operation when compared with service computation. In other words, database access is slower when compared with service invocation (note that we observe a local system, that is, we do not take a network, such as the Internet, into account). Even if we add more database servers, system usefulness will still be limited by the time it takes to execute a query. Contrary, if we add more Web services, the number of database servers limits the usefulness. The simplest reward model would be to award each state with $min\{i, j\}$ reward. This model is optimistic, as it does not take into account contention between services for database servers. If we consider that Web services may also compete for database servers, the following classical reward proposed in [22] may be used to describe the average number of busy servers:

$$r_{i,j} = m(1 - (1 - \frac{1}{m})^l)$$

where $l = min(i, j)$ and $m = max(i, j)$. This will result in state (3,2) having reward 15/9, state (2,2) having reward 3/2 and all other up-states having reward 1. If we solve this model in SHARPE for $\lambda_S = 1/720$ and $\lambda_D = 1/1440$, results obtained in Figures 3.31 and 3.32 are obtained.

Note significant discrepancy between reliability and expected reward curves in the interval approximately (0,1000), where the effect of rewards for states (3,2) and (2,2) is obvious. Even after first database server has failed and the system makes transition into (2,2), the reward is still bigger than one, compared to reward of one in case that one service failed and the system is in (3,1). After this, all rewards are equal to one, and the reward model quickly degenerates into reliability model, which is also evident in Figure 3.31, because two curves are afterwards almost fully overlapped. The cumulative expected reward, given in Figure 3.32, demonstrates that cumulative expected reward grows until the system fails. The mean time to failure is 1800 and can be calculated from the reliability model (Markov chain).

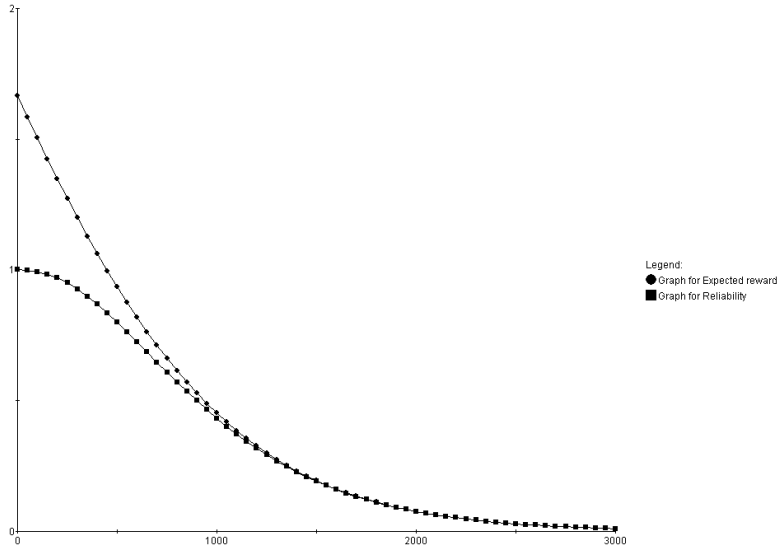

Figure 3.31: Comparison of transient expected reward and reliability

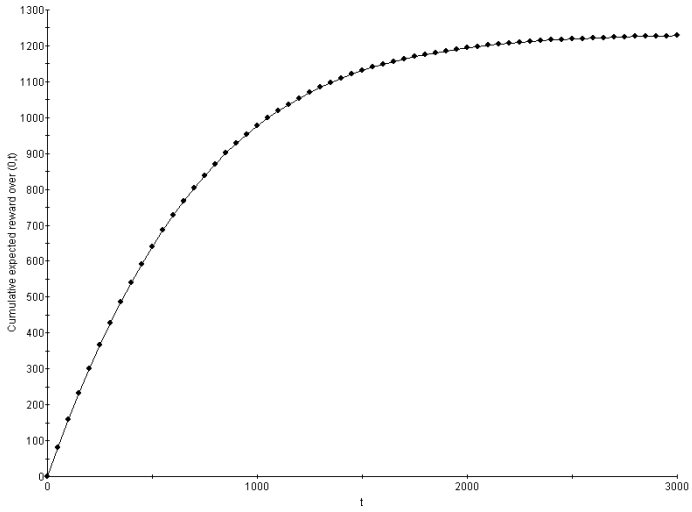

Figure 3.32: Expected cumulative reward

3.2 Qualitative Models

The main problem of analytical methods presented so far is the necessity to parameterize model elements. In other words, even after an arbitrary complex formal model has been structurally developed (e.g., combination of series and parallel configurations in an RBD or a set of states and transitions in a Markov chain), which corresponds to process or service description, it is necessary to assign parameters to models elements. Parameters, such as failure and repair rates as well as adequate distributions, can be in principle derived using historical data (past system behavior) or based on measurements. When

neither is possible, qualitative methods are used. Usage of qualitative methods to determine service and process availability is not the main focus of this work, however, for reasons of completeness, we will briefly survey main qualitative models which may be used to assess service availability.

Qualitative methods are based on reference models, which represent a meta-concept (not formally defined as a metamodel in the MDA context, but with similar semantics) that can be instantiated in different contexts and has the two main attributes: generality and advisory/recommendation character. Because of the second attribute, reference models are frequently called best-practice models. If a reference model is intended for process management, it is called a process model. When used in the assessment or compliance verification context, they are also frequently referred to as maturity models. Input into process models is usually structured in form of a questionnaire, which collects data about enterprise assets and process properties. Based on answers, consultant who is often backed up by a knowledge- or rule-based tool, makes an assessment and assigns a property class to the observed process (e.g., availability class). In most cases, however, availability is not explicitly modeled as a property, but can be derived as one of the risk or threat attributes/factors. In the following sections we will investigate most important process models for managing IT-structures and IT-processes with respect to process and service availability. More information about practical application of process models can be found in Chapter 4, Section B.2 "Qualitative and Process Management Tools".

3.2.1 CMMI

Capability Maturity Model Integration (CMMI) [198] is a quality management model intended for system and software development, proposed as an extension of well known Capability Maturity Model (CMM) [161]. The model is divided into CMMI for Services and CMMI for Acquisition. Only CMMI for Services (further CMMI) will be discussed here.

The basic CMMI philosophy is similar to ISO 9000 standard, but the model is not formally based on it because of relatively strong specialization towards system and software development. The main purpose of CMMI is to assess and improve quality of the product development process in organizations. This is done by analyzing strengths and weaknesses of the production process with the goal to suggest improvement measures. CMMI integrates in one process model many assessment aspects that were previously parts of different and incompatible models. This enables extension as well as integration of different disciplines (e.g., hardware/software co-design). The primary focus of CMMI is to suggest countermeasures for discovered weaknesses (e.g., security threat, single point of failure, documentation), but it can also be used to verify and certify the level of maturity with respect to some standard.

Process assessment with CMMI can be performed using a stepwise process, where for each process area a maturity level can be assigned: incomplete, performed, managed, defined, quantitatively managed and optimizing. CMMI supports 24 process areas: capacity and availability management, causal analysis and resolution (CAR), configuration management, decision analysis and resolu-

tion, integrated project management, incident resolution and prevention (IRP), measurement and analysis, organizational innovation and deployment (OID), organizational process definition, organizational process focus, organizational process performance, organizational training, project monitoring and control (PMC), project planning, process and product quality assurance, quantitative project management (QPM), requirements management, risk management (RSKM), supplier agreement management, service continuity (SCON), service delivery, service system development, service system transition and strategic service management.

Following process areas have been found to be relevant for availability assessment: the CAR policy identifies and systematically addresses root causes of defects and other problems; the IRP policy establishes an approach to incident resolution and prevention, to identifying, controlling, and addressing incidents and for selected incidents, determining workarounds or addressing underlying causes; the OID policy identifies and deploys process and technology improvements that contribute to meeting quality and process-performance objectives; the PMC policy monitors performance against the project plan and manages corrective action to closure when actual performance or results deviate significantly from the plan; the QPM policy manages quantitatively the project using quality and process-performance objectives and statistically manages selected subprocesses within the projects defined process; the RSKM policy defines a risk management strategy and identifies, analyzes, and mitigates risks; the SCON policy establishes a service continuity plan that enables resumption of key services following a significant disruption in service delivery, provides training in the execution of the plan, and verifies and validates the plan.

As can be seen, not a single policy mentions availability explicitly, but availability can be derived as an attribute of security or risk. Finally, SCON defines disruption of service delivery, which is comparable to the notion of a failure.

3.2.2 ITIL

ITIL (IT Infrastructure Library) [3] is a set of concepts and policies for managing information technology (IT) infrastructure, development and operations. Its development started in the 80s on the request of the British government by Central Computer & Telecommunication Agency, as an answer to the growing dependency and complexity of IT frameworks. Since 2000 it is being published as a set of books, each of which covers specific IT management topic, by the UK Office of Government Commerce.

ITIL2 comprises the following best-practice sets/policies: service delivery, service support, ICT infrastructure management, security management, business perspective, application management, and software asset management. After ITIL2 has been criticized as too complex and confusing, ITIL3 reduced the number of sets to service strategy, service design, service transition, service operation and continual service improvement. The following ITIL subsets are relevant for service availability: incident management, problem management, service level management, and availability management.

The goal of incident management is to restore normal service operation as quickly as possible and minimize the adverse effect on business operations, thus ensuring that the best possible levels of service quality and availability are maintained. Normal service operation is defined here as service operation within Service Level Agreement (SLA) limits. The goal of problem management is to resolve the root cause of incidents and thus to minimize the adverse impact of incidents and problems on business that are caused by errors within the IT infrastructure, and to prevent recurrence of incidents related to these errors. A problem is an unknown underlying cause of one or more incidents, and a known error is a problem that is successfully diagnosed and for which either a work-around or a permanent resolution has been identified. Problem management is different from incident management. The principal purpose of the problem management is to find and resolve the root cause of a problem and prevention of incidents; the purpose of incident management is to return the service to normal level as soon as possible, with smallest possible business impact. Service level management provides for continual identification, monitoring and review of the levels of IT services specified in the service level agreements (SLAs). It is connected with availability management, capacity management, incident management and problem management to ensure that required levels and quality of service are achieved. Finally, availability management allows organizations to sustain IT service availability in order to support the business at a justifiable cost. The high-level activities are: realize availability requirements, compile availability plan, monitor availability, and monitor maintenance obligations.

ITIL departs from the traditional view that business availability can be adequately represented as pure system component availability. The traditional IT approach to measurement and reporting provides an indicator on IT availability and component reliability which is important for the internal IT support organization. However, to the business and particularly end-user, these measures fail to reflect availability from their perspective and are rarely understood. This often fuels mistrust between the business and IT where despite periods of instability the availability target has been met even though significant business disruption has occurred and customer complaints have been received. Furthermore, this method of measurement and reporting can often hide the benefits delivered to the business from IT improvements. Traditional IT availability measures can simply mask real IT "added value" to the business operation. While traditional IT availability measurement and reporting methods can be considered appropriate for internal IT reporting, disadvantages of this approach are that they:

- fail to reflect IT availability as experienced by the business and the user
- can conceal service "hot spots" whereby regular reporting shows the SLA met, but the business and/or user is becoming increasingly dissatisfied with the actually provided service
- do not easily support continuous improvement opportunities to drive improvements that can benefit the business and the user

- can mask IT "value add" where tangible benefits to the business and user have been delivered but the method of measurement and reporting does not make this visible

According to ITIL, availability, when measured and reported to reflect the user experience, should provide a more representative view on overall IT service quality. The user view of availability is influenced by three factors: frequency of downtime, duration of downtime and scope of impact. Measurements and reporting of user availability should therefore embrace these factors. The methodology employed to reflect user availability could consider two approaches:

- Impact by user minutes lost: to base calculations on the duration of downtime multiplied by the number of users impacted. This can be the basis to report availability as lost user productivity or to calculate availability percentage from a user perspective.

- Impact by business transaction: to base calculations on the number of business transactions that could not be processed during the period of downtime. This provides a better indication of business impact reflecting differing transaction processing profiles across the time of day, week etc. In many instances it may be the case that the user impact correlates to a vital business function, e.g., if the user takes customer purchase orders and a vital business function is customer sales. This single measure is the basis to reflect impact to the business operation and user.

As can be seen, ITIL is much more concrete in availability treatment when compared with CMMI. Furthermore, it introduces an important availability aspect, user- or business-perceived availability, as opposed or rather complemented by IT-infrastructure availability. This consideration is included into our reference architecture model and availability definitions in Chapter 2.

3.2.3 CITIL

CMMI for IT Operations (CITIL = CMMI+ITIL) is the integration of ITIL into series of CMMI models. CITIL supports organizations that operate IT systems. The combination of ITIL and CMMI makes it possible to use CMMI's best practices also for an IT operations organization. Moreover, this integrated model covers the complete IT lifecycle with a common language and defined interfaces between operations and development. To cover the complete lifecycle, CITIL can be combined either with the CMMI for Development (CMMI-DEV)[59] or with the CMMI for Acquisition (CMMI-ACQ)[83].

CMMI-ITIL defines typical development steps to improve IT operations and their maturity levels. Through assessments, which are executed according to the official CMMI processes, it can be determined how well an organization executes practices of IT operations. As opposed to CMMI for Services, CMMI-ITIL is not a new model, instead it is integration of the established ITIL in CMMI with full conformity with ITIL and CMMI. In addition CMMI-ITIL deals specifically with IT operations while CMMI for Services covers all types of services and is therefore naturally much more general.

Similarly to ITIL, following areas can be found to be relevant for availability assessment: incident management, problem management, service level management, and availability management. However, as already explained, ITIL is constrained to IT-services only, making it more manageable.

3.2.4 ISO/IEC 15504 – SPICE

ISO/IEC 15504 also known as SPICE (Software Process Improvement and Capability dEtermination)[209] is a framework for process assessment developed by the joint technical subcommittee between ISO and IEC (International Electrotechnical Commission). ISO/IEC 15504 was initially derived from the process lifecycle standard ISO 12207, and the ideas of many maturity models like CMM.

ISO/IEC 15504 contains a reference model, which defines process dimension and capability dimension. The process dimension defines processes divided into the five categories: customer-supplier, engineering, supporting, management and organization. The most interesting parts, namely IT service process and enterprise process, are however still under development. For each process, a capability level on the following scale is defined: optimizing, predictable, established, managed, performed and incomplete. Capability of each process is measured using process attributes: process performance, performance management, work product management, process definition, process deployment, process measurement, process control, process innovation, process optimization. Each process attribute consists of one or more generic practices, which are further elaborated into practice indicators to aid assessment performance.

ISO/IEC 15504 is used in two contexts: process improvement or capability determination. Applied to dependability, the first context is equivalent to fault prevention, and the second to availability assessment. In the context of availability assessment, critical SPICE attributes are process performance, process measurement and process control. Unfortunately, the standard is relatively broad on these attributes, and no single process dimension accurately reflects service availability requirements. It is to be expected that part 8 of the standard (IT Service Management Process Assessment Model), which is currently in the ballot phase, will provide assessment criteria similar to those offered by the CMMI for Services.

3.2.5 CobiT

CobiT [105] stands for Control Objectives for Information and related Technology and is globally accepted framework for IT governance. It was introduced by the Information Systems Audit and Control Association (ISACA) and the IT Governance Institute (ITGI) in 1992. CobiT is constantly being revised and regularly published. The current (March 2009) version is 4.1.

CobiT assists the management and business process owners in their understanding and managing of risks and benefits associated with IT and related technology. CobiT ensures that IT supports business goals of the company, that IT investments are optimized, and that appropriate risk- and change-management

procedures exist. It also helps bridge the gaps among business requirements, control needs, and technical issues as a control model to meet the needs of IT governance and ensure the integrity of information and information systems.

In availability context, an important role of CobiT is the provision of metrics and maturity models to measure achievement of business and IT goals, evaluation of IT performance as well as enabling identification of responsibilities of business and IT process owners. The CobiT framework is subdivided into 4 domains and 34 processes. The four domains are the following:

- Plan and Organize. This domain refers to the strategy and tactics for IT contribution and the way that IT meets business objectives.
- Acquire and Implement. This domain deals with the realization of the IT and more precise with how solutions are being identified, developed, acquired or implemented, as well as how these solutions are integrated into business processes and systems are changed and maintained.
- Deliver and Support. This domain covers the actual delivery of required services, and in particular management of security and continuity, service support for users, and management of data and operational services.
- Monitor and Evaluate. Performance management, monitoring of internal control, regulatory compliance and governance are treated in this domain.

The CobiT framework is measurement-driven, which means that it provides models and metrics that enable evaluation of achievement of certain goals. Each of 34 processes can be rated from the maturity level of non existent (0) to optimized (5). Specifically, following criteria are relevant in the context of this work: confidentiality, availability, integrity, and reliability. Maturity levels are not meant to be used as a threshold model in the sense that one cannot move to the next level before having met all conditions of the previous level. Therefore, it can occur that parts of one process are assigned to different maturity levels as they fulfill requirements of different maturity levels. The CobiT framework document provides descriptions of different maturity levels for each process, naming specific actions and resulting states that characterize a process in the corresponding maturity level. Even though the descriptions are process-specific, there are common characteristics in maturity levels across processes.

Based on the exemplary CobiT evaluation performed within a student project [25], we identified the following concrete CobitT processes as relevant for availability assessment: PO1 (Define a Strategic IT Plan and direction), PO2 (Define the Information Architecture), PO3 (Determine Technological Direction), PO4 (Define the IT Processes, Organization and Relationships), PO5 (Manage the IT Investment), PO6 (Communicate Management Aims and Direction), PO7 (Manage IT Human Resources), PO8 (Manage Quality), PO9 (Assess and Manage IT Risks), ME1 (Monitor and Evaluate IT Processes), DS1 (Define and Manage Service Levels), DS2 (Manage Third-party Services), DS3 (Manage Performance and Capacity), DS4 (Ensure Continuous Service), DS5 (Ensure Systems Security), DS6 (Identify and Allocate Costs), DS7 (Educate and Train Users) and DS12 (Manage the Physical Environment).

During the assessment we also experienced significant difficulties in obtaining unanimous assignment of maturity levels, as this process is still very subjective and depends on the consultant performing the assessment, as well as on the willingness and interest of interviewed personnel to answer the questionnaire correctly and in-depth.

3.2.6 MOF

Microsoft Operations Framework (MOF)[144] consists of integrated best practices, principles, and activities that provide guidelines for achieving reliability for IT solutions and services. MOF provides question-based guidance that allows to determine current organization requirements, as well as plan and assess the future activities.

The guidance in Microsoft Operations Framework encompasses all activities and processes involved in managing an IT service: its conception, development, operation, maintenance, and ultimately its retirement. MOF organizes these activities and processes into service management functions (SMF), which are grouped together in phases that mirror the IT service lifecycle. Each SMF is anchored within a lifecycle phase and contains a unique set of goals and outcomes supporting objectives of that phase. An IT services readiness to move from one phase to the next is confirmed by management reviews, which ensure that goals are achieved in appropriate fashion and that IT goals are aligned with the organization's goals.

The IT service lifecycle is composed of three ongoing phases and one foundational layer that operates throughout all other phases: plan phase, deliver phase, operate phase and the manage layer.

Reliability SMF belongs to the plan phase of the MOF IT service lifecycle. It addresses creating plans for following attributes: confidentiality, integrity, availability, continuity and capacity. Ensuring reliability involves three high-level processes:

- Planning: gathering and translating business requirements into IT measures
- Implementation: building of various plans and ensuring that they can meet expectations
- Monitoring and improvement: proactively monitoring and managing the plans and making necessary adjustments

Reliability is naturally connected with the business/IT alignment, therefore many outputs of reliability SMF, such as availability plan, capacity plan, data security plan, and monitoring plan, provide input for the activities described in the business/IT alignment SMF. Basic outcomes of reliability SMF are that IT capacity is aligned to business needs, services are available to users when needed, critical business services are available even during significant failures, and data integrity and confidentiality maintained.

In the planning phase, service reliability requirements are defined. For this purpose projected SLAs, regulations and laws, internal policies, risk and impact

analysis, future change plans, projected operating level agreements, service dependencies and reliability parameters of each service are used. Based on this input, SLA is defined in terms of hours of operation, maintenance windows, recoverability time frames, continuity requirements, etc., as well as data classification and corresponding data handling policy, service priority and growth plans. After taking into consideration service map, historical information, technical information and operating guides, a comprehensive reliability specification model and templates are generated as the final output of this phase.

In the implementation phase, reliability management involves following activities:

- Developing various plans: availability, capacity, data security, disaster recovery, monitoring
- Reviewing and adjusting plans for suitability before approving them

Finally, the third process of reliability management is monitoring and improving plans, an ongoing procedure that ensures that the first two processes have been followed, that metrics are reported on, that exceptions to targets are tracked, and that improvements are fed back into the plan phase. Proper monitoring ensures that either the original objectives are being achieved or steps are being taken to improve reliability or adjust business expectations. This process includes following activities:

- Monitor service reliability
- Report and analyze trends in service reliability
- Review reliability

Monitoring activity uses a monitoring plan and data feeds to produce reliability specific metrics and management dashboard. They are used together with service availability reports, MTTR reports, problem management reports and original business SLAs to report and analyze reliability trends in form of reliability reports and recommendations for improvement or technical evaluation, requirement review, improved or reduced service level commitments. The last phase of reliability management is the business review (how trend data affect SLAs), generation of prioritized and lists of improvement recommendations and RFCs to implement improvement activities and data storage for subsequent trend analysis.

3.2.7 MITO

Maturity Model for IT Operations (MITO) [180] has been developed by the Swiss Association for Quality Promotion (SAQ SG). Its goal it similar to CITIL, namely, to provide process management and maturity model for IT operations only, excluding other process types. It is also partially derived from ITIL. MITO divides IT operations into five process classes: customer services, engineering services, management services, strategic services and internal services.

We will discuss only those processes and sub-processes that are directly related to service availability assessment. In the customer services group, providing services represent the actual production, that is, running programs or making backups. Issue management treats incident reports, based on the monitoring service provision which observes and measures service provision in order to prevent failures or to raise issues for the issue management if a failure could not have been prevented. Security of provided services has similar role, focused on security instead on fault-tolerance.

The required infrastructure and services provided on that infrastructure are developed and managed by engineering services. Problem management process supports management of reported incidents, together with the incident processing. The configuration management controls modification of the hardware and/or software configuration as a results of the incident.

The first two groups of processes are rather technical. Management services control both processes with the focus on the SLA management. The purpose of process management of service level agreements is to listen to the customers and monitor service provisioning. It takes care of needs and requirements of the customers as well as of assignment of services contracted in SLAs to different service centers, with the goal to fulfill and guarantee the contracted SLA. Resources of each service center are managed by the management of operation level agreements (OLA) process. It provides input for the forecast, capacity and service planning, whose goal is to ensure the necessary infrastructure and resources which will meet both SLA and OLA.

Finally, the internal services process provides measurement and evaluation. All required inputs for the SLA determination are provided by the service level controlling process which also indicates to the management of SLA eventual deviations from requirements. Measurements are carried out on the level of OLA and combined for the SLA.

MITO defines five levels of maturity: stochastic, repeatable, tracked, measured, optimized. The maturity level is determined by the assessment of all processes. To every process, one of the following quality stages is assigned: intuitive, planned, applied, analysed and controlled. Then, process quality stages are mapped to maturity levels.

3.2.8 ISO/IEC 27002

ISO/IEC 27002 standard, renumerated from ISO/IEC 17799 on which CobiT is based in 2007, is the information security standard published jointly by the ISO and the International Electrotechnical Commission (IEC). Its full name is Information technology - Security techniques - Code of practice for information security management, and it represents an evolution of the British Standard BS-7799. ISO/IEC 27002 provides best practice recommendations on information security management for initiating, implementing or maintaining information security management systems (ISMS). Information security is defined within the standard as preservation of confidentiality (ensuring that information is accessible only to those authorized to have access), integrity (safeguarding the accuracy and completeness of information and processing methods) and avail-

ability (ensuring that authorized users have access to information and associated assets when required).

The standard defines following main processes: risk assessment, security policy, organization of information security, asset management, human resources security, physical and environmental security, communications and operations management, access control, information systems acquisition, development and maintenance, information security incident management, business continuity management and compliance.

Within each section, information security controls and their objectives are specified and outlined. The information security controls are generally regarded as best practice means of achieving those objectives. For each control, implementation guidance is provided.

3.2.9 ISO 12207/IEEE 12207

ISO 12207 is the standard for software lifecycle processes. It defines all tasks required for developing and maintaining software. It establishes a software lifecycle process, including processes and activities applied during the acquisition and configuration of system services. Each process has a set of outcomes associated with it. There are 23 processes, 95 activities, 325 tasks and 224 outcomes.

The main objective of the standard is supplying a common structure so that the buyers, suppliers, developers, maintainers, operators, managers and technicians involved with software development use a common language. This common language is established in the form of well defined processes. The structure of the standard was intended to be conceived in a flexible, modular way so as to be adaptable to the necessities of whoever uses it. The standard is based on two basic principles: modularity and responsibility. Modularity means processes with minimum coupling and maximum cohesion. Responsibility means to establish responsibility for each process, facilitating application of the standard in projects where many people can be legally involved.

The set of processes, activities and tasks can be adapted according to the software project. These processes are classified in three types: basic, support and organizational. The support and organizational processes must exist independently of the organization and the project being executed. Basic processes are instantiated according to the situation.

The standard itself is part of many frameworks and other standards, such as Rational Unified Process (RUP) or IEEE 12207. The latter is a standard that establishes a common framework for software life cycle process. It defines a set of processes, which are in turn defined in terms of activities. The activities are broken down into a set of tasks. The processes are defined in three broad categories primary life cycle processes, supporting life cycle processes and organizational life cycle processes. This structure closely mirrors ISO 12207. The primary life cycle processes are acquisition, supply, development, operation and maintenance. The supporting life cycle processes are audit, configuration management, joint review, documentation, quality assurance, problem solving, verification and validation. Finally, the organizational processes are management, infrastructure, improvement and training. Several sub-processes are relevant for

availability assessment, namely operation and maintenance (primary life cycle, ICT infrastructure level), audit, configuration management, quality assurance, problem solving, verification and validation (supporting life cycle), and finally the infrastructure process (organizational).

3.2.10 Relationships between Maturity Models in the Availability Context

It can be summarized that reference models and IT standards cover three broad aspects of IT organization and corresponding life cycle: IT management, IT development and IT operations.

Further, each aspect can be focused on:

- Process definition: best-practice recommendation on optimal process identification and specification.
- Process requirements: systematic assessment procedure, usually described as asset description and evaluation, which results in the categorization into "compliant" or "non-compliant" group.
- Process improvement: system oriented process, offers suggestions, actions and countermeasures to improve critical process aspects, parts or attributes.

The coverage of described reference models is given in Figure 3.33 (this Figure, as well as Figure 3.34, is adapted from [88]). CobiT provides assessment for all aspects, but focuses mostly on process requirements. ITIL on the other hand, plays the role of a framework which integrates other standards (e.g., ISO 12207 and ISO 27002) whose focus are IT-operations and IT-development. It does not cover IT-management, but it references CobiT for that purpose (see Figure 3.34). ITIL also references CMMI for the purpose of process requirements and improvement. CobiT is explicitly based (references) on ISO 27002, but also extends its IT-management capabilities. ITIL and MOF are partially based on ISO 12207. While CobiT is attempting to cover the aspect dimension as completely as possible, MOF attempts to cover the focus in its entire scope.

Figure 3.34 depicts dependencies between reference/maturity models and standards. It is obvious that ITIL and CMM influence most of the other models, as all of them use parts of ITIL and/or CMM. This is partially the consequence of historical development of maturity models. The ordering of shapes in Figure 3.34 tries to illustrate this, as the time axis (not shown) is approximately vertical, with time increasing in the downwards direction.

Figure 3.34 also explains why ITIL or CobiT certification is not possible. Both are not standards, but collections of best practices based on standards such as ISO 27002 or ISO 12207. Therefore, only certain generic but standardized reference model aspects can be certified under ITIL or CobiT. This is not true only for ITIL and CobiT, but for most of the other maturity models (e.g., CMMI is based partly on ISO 9001, which is not shown here). Due to deep interconnections, most of them also provide compatible availability assessment

mechanisms using key performance indicators (KPI) and key goal indicators (KGI), such as those described for each model in the previous sections.

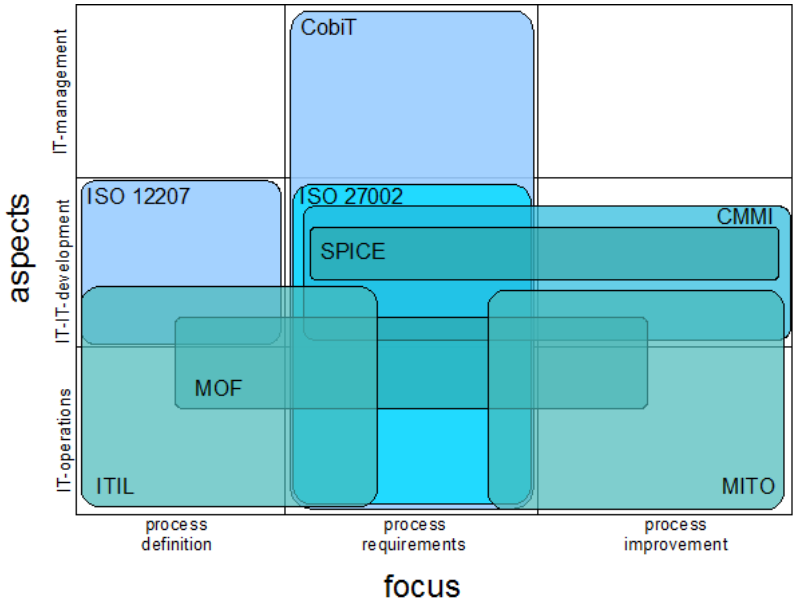

Figure 3.33: Coverage of reference/maturity models and standards

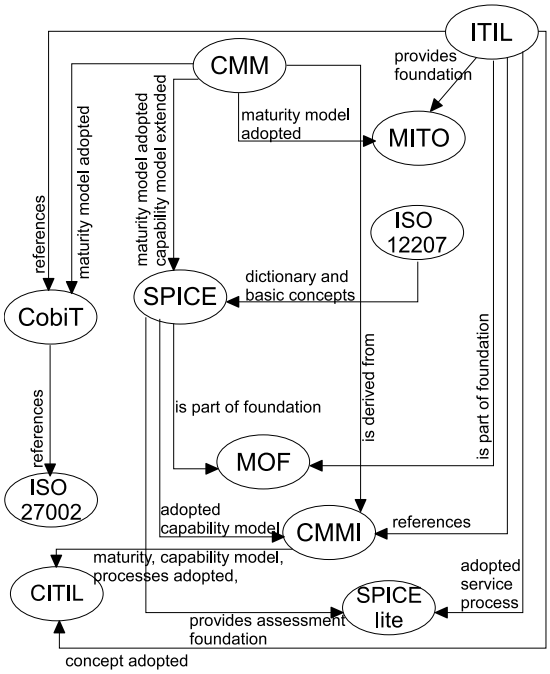

Figure 3.34: Dependencies of reference/maturity models and standards

Chapter 4

Tools for Availability Assessment

In this chapter, the most important tools for availability assessment will be compared. They will be divided in following groups: quantitative tools (analytic and simulation; benchmarking), qualitative tools (risk assessment; process management) and hybrid tools that use both qualitative and quantitative methods for availability assessment. Interoperability and general usability of all tools will be discussed. The historical perspective of availability assessment tool development is given in Figures 4.2 and 4.3. Finally, summary of the survey results is given in Table 4.1. Detailed information about all surveyed tools can be found in Appendix B. To enable quick reference, for each tool in Table 4.1, a page number reference to Appendix B is given, where complete tool description can be found.

All tools were investigated as well as described in detail in Appendix B following this structure:

- Source/Reference: available material about the tool, including scientific papers, documentation, standards, manuals and Web resources.
- Project status: describes the status of the tool development, indicating whether there is continuous support for the tool.
- License type: lists all known license types, including commercial, academic, test and evaluation.
- General purpose: short overview of the main functions, supported models and solution methods and outputs.
- Platform: lists available platforms that are supported by the tool. Additional infrastructure requirements (e.g., a database in the background for persistence or an office suite for report generation) are also listed.
- Model:
  - Model class: associates the tool with one (or more) classes: quantitative (analytical, simulation, benchmarking) or qualitative. The level

at which the tool operates is also listed (IT-infrastructure, service or process).

  - Model types: lists the model types supported by the tool (e.g., Markov chain, Petri net, questionnaire, workflow, etc.).
  - Model description: short overview of the model used by the tool and its properties.
  - Systems modeled: describes which system classes can be modeled using the tool (e.g., repairable systems, computer networks, software, etc.).
  - Model input: lists required model inputs (e.g., graph vertices and edges, failure modes, etc.).
  - Model output: results of the model analysis (e.g., availability metrics, MTTF, MTTR, etc.).
  - Interfaces: describes technical interfaces of a tool:
    * Input interfaces: lists available input mechanisms of a tool.
    * Output interfaces: lists available methods for output and report generation/export.

- Use cases: lists known use cases where the tool has been applied for availability analysis.

- Assumptions and restrictions: lists possible factors that influence the tool exploitation, such as scalability, speed or security.

The content of this chapter represents the continuation of the study reported in [112] as well as [138].

4.1 Quantitative Tools

The general purpose of quantitative tools is to perform quantifiable availability assessment of the target system, through rigorous modeling, and to solve the model using analytical methods or simulation. A separate group are benchmarking tools, which use on-line empirical methods to determine the quantifiable amount of system's availability through evaluation of predefined availability indicators.

4.1.1 Analytical and Simulation Tools

Design and implementation of analytical and simulation tools is relatively streamlined in terms of input and output, as most of them are based on the concepts pioneered by various academic tools developed in the 70s and 80s (e.g., ARIES, GRAMP/GRAMS, SURF, METASAN, HARP etc.). The common property linking them is strong support for the static, off-line, formal modeling of fault tolerant systems. For that purpose, a variety of methods and models are supported, such as Markov chains, Petri nets, reliability block diagrams, fault trees, stochastic activity networks, etc. Almost all tools support at least one of these

standard models. The tool output is also more or less similar, covering various steady-state and transient availability, reliability and performability metrics (these are detailed in descriptions of the respective tools).

Among the tools surveyed in this category, the following were found to be relevant for further consideration in the context of service and business process availability assessment: Isograph Toolbox (FaultTree+, AvSim+, Reliability Workbench, NAP, AttackTree+), METFAC, Möbius, OpenSesame, Reliass, Sharpe 2002, SPNP, Figaro, BQR and Mathworks Stateflow.

IsoGraph and Sharpe 2002 are powerful toolboxes that combine many models within one framework. However, models cannot be interchanged, that is, model transformation has to be performed manually. IsoGraph also offers rudimentary failure prediction, which is static and based on predefined failure rate data and protocols, primarily for the electrical/electronic devices. Furthermore, it explicitly supports modeling of communication networks and attacks, which is possible only implicitly with Sharpe 2002. Sharpe 2002 is closely coupled with SPNP, which is a Petri net-based tools offering various reward models. Together, they form a coherent platform that can be compared to IsoGraph in complexity and power. Both offer analytical and simulation (Monte Carlo) solving methods. It should be further mentioned that IsoGraph is a commercial tool, while Sharpe 2002 and SPNP are academic efforts. METFAC is another example of a successful academic project, which is, however, limited to Markov models only, and as such cannot be considered equal in expressive modeling power to IsoGraph or Sharpe/SPNP combination.

Möbius is another successful academic tool which combines different models (e.g., stochastic queues, Petri nets, fault trees etc.). The difference when compared to IsoGraph and Sharpe is that Möbius allows different problem aspects to be described using different models that can be integrated and combined. It also solves the models using analytical and simulation techniques. The issue of model combination is very important reference point for our further study. In the following chapter, the model integration idea will be examined in more detail.

OpenSesame tool is based on limited model transformation capabilities, namely, reliability block diagram is given as input, and it is then internally transformed into a stochastic Petri net. While this process is admittedly automatized, it does not allow to model different problem domains using different models.

Reliass is another example of an integrated product where availability assessment is not the only (or primary) target. This tool consists of the number of modules, but only some of them enable modeling of fault-tolerant systems using reliability block diagrams and Markov models. Other modules use this (limited) availability information to perform other tasks, such as management and various kinds of vulnerability analysis. This approach is very similar to the approach that qualitative-based tools have in the area of process management and optimization. Reliass however, concentrates only on the IT infrastructure level, and not on the process/service level.

BQR Care tool is part of the larger toolbox, and specializes in Failure Mode, Effects and Criticality Analysis (FMEA/CA), as well as fault tree and reliability

block diagram analysis. Criticality analysis is limited to the IT level, actually only to predefined libraries of electronics and mechatronics (electro-mechanical) parts.

In this context, Mathworks Stateflow is positioned as a general purpose simulation environment that can be used for modeling and simulation of fault-tolerant systems. It does not offer any availability-specific extensions, but availability-relevant metrics can be simulated using statecharts, finite state machines or temporal logic. Stateflow simulator is very capable and advanced with respect to the number of states, complexity, speed, and solution quality, when compared with simulators offered by other tools.

Finally, the last relevant tool in this group is FIGARO, which offers a novel modeling approach: the observed system is modeled using abstract object-oriented FIGARO language, which is then transferred into the model that best fits the given description (e.g., Petri net, Markov model, reliability block diagram, or fault tree). This process is based on the knowledge/rule base and is completely transparent to the user. FIGARO is coupled with KB3 Workbench, which accepts models generated from FIGARO language and then solves them (analytically or using simulation methods). This approach is very important as it does not require intimate knowledge of complex modeling languages and formalisms. Instead, it is enough to understand the FIGARO description language.

Analytic and simulation tools offer very powerful mechanisms for modeling availability properties of fault-tolerant systems. The major issues that were discovered during this study are:

- *Scalability*: managing and solving increasing number of states in a static modeling procedure is highly problematic. It is not only the issue of performance and resources (although this can also be a limiting factor), but of manageability and involvement of the human factor. As the hardware/software systems grow, it is simply unrealistic to expect that classical modeling approach, where each system element has to be individually modeled and described, can be effectively scaled. Even with the use of modern graphical user interfaces that most of the tools provide, it is very difficult to develop and maintain models that are comparable in scale to the real-world problems. This problem is only exacerbated in SOA systems, which are characterized by rapid changes and fast-paced dynamics.

- *Steady-state metrics*: most of the tools offer only metrics that are based on "off-line" modeling methods, which means that model elements are parameterized using historical/statistical data. As such, models of complex IT systems tend to be either too simple or too imprecise. The value of availability assessment data obtained in that way is highly questionable.

- *Knowledge required*: all tools (except FIGARO) require very detailed and high-level, professional knowledge of modeling methods (e.g., Markov chains, Petri nets, finite state machines, etc.). This proves to be a barrier

for widespread utilization and industry penetration (with the exception of mission critical systems).

- *Model integration*: available models are very rich, but mostly separated. It is frequently impossible (with the exception of Möbius) to model each problem domain using a separate modeling method and to integrate all models thus obtained. Even Möbius does not allow for hierarchical, but only flat model composition.

- *Abstraction level*: all mentioned tools are very technically- and IT-oriented. None of them explicitly provides capabilities or facilitates availability modeling and assessment at the business process/service level.

- *Purpose*: The tools were created primarily for the purpose of system development: their goal is to give insight into the future behavior of the system. If they are to be used for availability evaluation of the existing systems, reverse engineering must be performed to build system models. None is capable of automatic/semi-automatic model generation from an existing system description.

4.1.2 Benchmarking Tools

Contrary to off-line, static modeling tools, benchmarking tools perform on-line analysis of system availability using empirical and test-based process. The goal of this part of the survey was to discover those tools that explicitly support quantifiable availability measurements, as opposed to static, off-line modeling of system availability. The relevant tools identified in this category are: Eclipse Test and Performance Tools Platform (TPTP), Exaustif, FAIL-FCI, Networked Fault Tolerance and Performance Evaluator (NFTAPE), and Quake.

Eclipse TPTP is by far the most complex tool in this group. However, it is not explicitly oriented to availability assessment and measurement. Instead, it provides a generic framework within which measurement- and test-based tools can be further developed. Even so, native TPTP services, such as testing, tracing, profiling and monitoring, can be utilized to evaluate availability attributes of complex software systems. Eclipse TPTP does not support development of tools for hardware or service-level availability assessment. Another important feature of Eclipse TPTP in the context of our study is the support for software instrumentation.

Other tools categorized in this group are based on failure injection and measurement of its effects. In that way, they directly support empirical availability assessment. Exaustif is based on software implemented failure injection, where failures are injected into the system during various development phases to determine its availability and reliability. Being a black-box approach, this method is currently available in very controlled conditions only, namely for RTEMS/ERC32 and RTEMS/Intel (which are special purpose real-time operating systems for multiprocessor systems). FAIL-FCI is used to inject failures into grid systems. Failures are described using special FAIL language, which is

then translated to the C++ code, compiled, deployed and executed on different grid nodes. That way, different outages can be simulated and their impact to the overall grid availability measured. General methodology (e.g., for fine grained systems) does not exist. NFTAPE is similar in conception, only geared to communication networks. Finally, Quake injects failures into Web Service systems and enables on-line measurements of outages, load or integrity.

The following conclusions can be reached about the potential usage of benchmarking tools for service availability assessment:

- Tools are either very broad and of general purpose (e.g., Eclipse TPTP platform), or very narrow in scope (e.g., communication networks or real time operating systems). To our best knowledge, there is no general purpose availability-measurement/benchmarking on-line tool, which can be applied to assessing service-level availability.

- The tools we identified are the only on-line methods for availability assessment that have been discovered, and they are all based on failure injection methods. Failure descriptions have to be prepared in advance and used as input for the benchmarking system. There is no possibility to select relevant variables of a black box automatically and infer availability information using the parameterized model. No variable (parameter) selection procedure (method) is offered either. That means that very intimate system knowledge is required in advance in order to specify meaningful failure descriptions. This seriously limits tool applicability for dynamic SOA systems.

4.2 Qualitative Tools

Tools that are based on qualitative methods use descriptive, tabular and discrete evaluation techniques (such as audits, log files, interviews, questionnaires, implementation templates, etc.) to assess system availability. They can be coarsely divided into two subgroups: tools for risk management and for process management.

4.2.1 Risk Management Tools

Tools for risk management evaluate risk at the process level using qualitative methods. The relevant tools that were discovered are: ClearPriority, Secura 17799, COBRA, CounterMeasures, CRAMM, EBIOS, GSTOOL, ISAMM, OCTAVE, PILAR, PROTEUS, RA2 and RiskWatch. Most of them are based on international standards for risk management, such as ISO-27002 (formerly ISO-17799), ISO 27001, US-NIST-800-26-Standards, or COBIT-4.0. These tools use multi-phase process model, where the context is modeled first as a pre-condition (e.g., process description, available assets), then availability or security threats are given, followed by additional threat analysis. After this step, the ranking is usually performed, followed by risk management recommendation. Possible options in this step are risk reduction through restructuring, or risk transfer.

The process is terminated by consolidation of the IT-security concept. For all tools that were investigated, the following can be summarized:

- Availability is not explicitly mentioned as risk property, part of the risk or risk consequence.
- All tools are exclusively qualitative, and based on various tabular structures filled by informal description methods. The running system is never explicitly modeled and data monitoring/collection systems are not used.
- All proposed recommendations for risk mitigation/availability improvement remain in the area of general and informal design templates and patterns.
- Interpretation of the analysis results is highly questionable and depends heavily on the consultant performing the assessment.

4.2.2 Process Management Tools

Process management tools are complex software systems that are used to govern, maintain and optimize hardware and software infrastructures deployed within a given context. To do so, they use a mixture of formal and informal methodologies, such as workflow modeling on one side, and metadata repositories or threat/incident analysis on the other side. However, when dealing with availability, majority of methods that are used are pure qualitative methods, therefore these tools are classified in the qualitative group. Partial exception is HP Mercury, which allows for explicit runtime availability monitoring, and can therefore be also classified in the on-line benchmarking group. The exemplary tools in this group are: HP Mercury, Software AG CentraSite, Fujitsu Interstage Process Manager, IBM Tivoli Availability Process Manager and IBM High Availability Services.

HP Mercury, Interstage Process Manager and IBM Tivoli Availability Process Manager are environments with the similar goal: to offer complete management of the business process level from the governance and optimization perspective. They cover the entire process lifecycle: process modeling, integration, automation, deployment, management, maintenance and optimization. All tools offer standard connection between business process and IT-level, by analysing availability, risk and security parameters at the business process-level and transferring results into optimal investments into IT-infrastructure elements. The tools differ in the way business process is described and annotated. Interstage uses formal description methods known from the theory of business process modeling (e.g., Business Process Modeling Notation), whereas IBM Tivoli uses Component Failure Impact Analysis (CFIA) and availability (incident) management, as specified by ITIL. HP Mercury integrates repository model (configuration management database) with ITIL guidelines and workflow management and enactment engine. However, as IBM Tivoli is not meant to be used alone, but rather together with a suite of IBM process management tools (including a workflow enactment engine such as one included into Websphere), it can be said

that both tools exhibit similar capabilities and are of comparable complexity. The same holds for Mercury, which was recently acquired by HP and positioned within a range of HP process management tools (HP OpenView).

It is obvious that, in order to use any of these tools, very complex supporting infrastructure has to be in place, usually offered by the same company. This is not the case with IBM High Availability Services, which is not a product but a consulting service. It is mentioned here as example of a pure qualitative availability evaluation at the following levels of abstraction: business, data and event. This service analyses threats at the afore mentioned levels using questionnaire-based qualitative methodology and proposes optimizations to improve availability or security. More details are given in the tool description.

CentraSite differs from previously mentioned tools and services, as it represents an open framework for development of SOA governance solutions. No additional infrastructure is necessary, apart from the CentraSite-compliant directory. The idea behind this multi-vendor initiative is to eliminate the risk of a vendor lock-in, by providing an open, Web service-based infrastructure for process and service description. CentraSite builds on the idea of an open repository by defining procedures for policies, change control, dependency analysis, monitoring and reporting at both process- and IT-level. It is based on the qualitative analysis that has foundations in semantically enriched service directory. Several tools are available that are based on the CentraSite architecture, which optimize process availability (e.g., Interstage Process Manager).

Following are the summarized results of the above analysis:

- Process management tools are complex and require dedicated investments into management and governance infrastructure (except in case of CentraSite which is free and open, but still requires significant know-how).

- All tools connect the business layer with the IT layer, however, this connection is difficult to reuse and/or export, as it is based on proprietary tools/protocols and not on open standards (e.g., SOAP/ WSDL). Again, CentraSite is an exception of this claim.

- Qualitative methodologies and processes used are usually ITIL-compliant, although this is not always clearly documented or obvious.

- Basic idea of process managers is the definition of key performance indicators (KPI). Availability can be supported as one KPI, but there is seldom explicit support for it. Usually, performance metrics (such as transaction throughput) are supported by default.

- Process models are segmented (e.g., Tivoli uses proprietary Tivoli Unified Process), therefore tools are not mutually interoperable.

4.3 Hybrid Tools

Hybrid tools combine quantitative and qualitative methodologies for the purpose of availability assessment. Relevant examples that offer this kind of analy-

sis are: Relex Reliability Studio, Reliability Center Proact and LEAP, Reliasoft and PENPET.

Relex Reliability Studio combines Markov models and reliability block diagrams (quantitative) with risk and criticality analysis (qualitative) in order to perform failure impact analysis. However, while it is possible to connect two quantitative methods (e.g., Markov chain and RBD) and two qualitative methods (criticality matrix and risk assessment calculation), connection between qualitative and quantitative methodologies is not possible. For example, a Markov model cannot be associated with a workflow.

Reliability Center Proact and LEAP tools are two separate tools sold in one package. Proact performs root-cause-analysis, which is the quantitative method, while LEAP performs FMEA and opportunity analysis, which is the qualitative method. This tool actually enables connection between two models where LEAP can give ranking of unwanted events/results for the process described with Proact. This support is, however, limited to given results and is not general.

Reliasoft also claims to cover both qualitative and quantitative methods, but in reality the support is very rudimentary. Free-text descriptions coming from FMEA/FMECA analysis can be transformed (manually) into a flow chart model, which can be simulated using logical gates and memory.

PENPET is a tool that offers novel, chemistry-inspired modeling approach. The basic idea is to describe high level models as clusters of low level models. High-level properties are analysed qualitatively while low level properties are analysed quantitatively. For high level description, so called structural formulas are used (macromolecules). Smaller units (molecules or atoms) are described using generalized stochastic Petri nets. For molecules or atoms, failure rates or failure coverage are modeled in details, while macromolecules are described as qualitative combinations of atoms or molecules. Petri nets are transformed into Markov chains during the analysis.

The following conclusions have been made for the hybrid tools group:

- Although claiming to support both qualitative and quantitative methods, majority of investigated tools actually just offer two tools in one: there is no real connection between the modeling and analysis approach. The exception is PENPET tool which offers true hierarchical model integration.

- Support for qualitative availability assessment is, similar to pure qualitative tools, implicit and based on the risk factor analysis and failure impact and criticality analysis.

4.4 Interoperability and Usability

The issue of tool interoperability is very important for the availability assessment of complex SOA systems, as it may frequently be the case that one tool cannot model the entire system. For that reason it is necessary to investigate the options for model and/or result exchange between various tools.

As there is no accepted standard for exchange of availability models or evaluation results, possibilities for interoperability are very limited. At best, the surveyed tools provide some degree of compatibility with popular office suites (e.g., Microsoft Office), and even then, only as additional export format that can be further processed for presentation purposes. Limited number of tools support import of Microsoft Visio diagrams, while none of them supports import or export of UML models.

Usability is well addressed by the newer commercial and academic tools. All of them offer rich, albeit sometimes immature (e.g., Sharpe), user interface clients, which enable fast modeling and wizard-based editing of model element properties. Many tools have powerful report generators (e.g., Isograph Report Generator which is shared by all tools from the same family), enabling textual, tabular, graph and bitmap result manipulation. This represents a big step forward from the previous generation of availability assessment tools, that usually offered only textual/tabular (batch) I/O capabilities.

The focus and method of most important tools are given in Figure 4.1. Two main tool clusters are visible: quantitative hardware/software availability assessment tools and qualitative service/process management tools. It is evident from the figure that very few tools address quantitative (analytical) service availability assessment and vice versa, that (to our knowledge) almost no tool enables qualitative software/hardware availablity assessment.

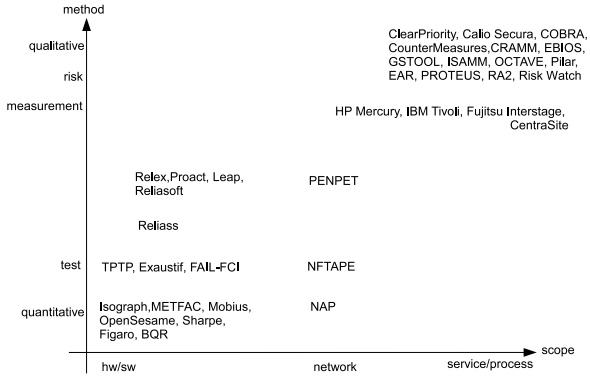

Figure 4.1: Focus and methods of availability assessment tools

4.5 Tool Comparison Summary

Based on the considerations and analysis given in previous sections, the following conclusions/goals have been formulated:

1. *Integration of quantitative and qualitative methods.* Several tools claim to integrate quantitative and qualitative methods. The level of integration, however, varies, from no real integration (separated models) to complete integration, albeit in a very narrow application domain (e.g., real-time

embedded systems). The problem of integration of qualitative and quantitative methods for availability-assessment remains unsolved in the general case.

2. *Integration of IT- and business process/service-levels.* This problem can be formulated as follows: how can availability assessment (or requirement) at the business process level be mapped to the underlying IT infrastructure and vice versa? In other words, how to invest in IT infrastructure in the optimal way, to improve the fulfillment of business process goals. Almost all tools from the process management category claim to have addressed/solved this problem. The truth is, however, that connecting business process and IT levels requires significant additional investments in software from the same vendor - leading to partial or total vendor lock-in. In such a closed domain, it is arguably possible to perform static mapping between the two levels. On-line (dynamic) mapping is not possible, as process enactment engines still lack the capabilities to perform runtime transformations. HP Mercury has to be partially excluded from this statement, as it allows network monitoring and mapping of results to the business process. CentraSite aims to provide an open environment for service (process-level) description as viable alternative. For the time being, however, the framework is limited to systems that expose their business operations and services as SOAP-based Web Services.

3. *Scalability/complexity.* Analytical and most of the process management tools rely on the static system description. This requires complex user interfaces and dedicated and intimate human involvement: an operator must describe all system states and availability properties in minute details. With the complexity of modern SOA systems growing, this is becoming non-feasible, as the process does not scale and is error prone. Furthermore, SOA systems are far from static, therefore, complex models must be manually maintained and updated whenever a change at the IT- or service-level is introduced, which happens very frequently. Not to underestimate is the problem of model solving and resources required for it, although this issue is becoming less relevant with hardware resources becoming increasingly available. Therefore, a new approach needs to be devised that departs from the standard white box approach, where all details of all system components are meticulously specified and modeled. Standard modeling approaches, developed for static and rarely changing mission critical systems, do not fit well to dynamic SOA systems.

4. *Steady state availability.* Connected with the previous point, majority of tools is thus limited to meaningfully assessing steady state service availability only.

5. *On-line and dynamic approaches.* Also related is the issue of on-line (dynamic) availability assessment, which uses the black box approach. The basic problem is that a SOA system should be observed as a black box that can be instrumented. The first step to solve is variable selection:

which of many instrumented system parameters are relevant for availability assessment? After that, appropriate parameterized models have to be generated that accept selected variables and assess system availability on-demand, on-line and dynamic. Among the surveyed tools, only benchmarking group offers rudimentary on-line capabilities, that are restricted to evaluating test cases on non-production systems. Therefore, a model that is dynamic and black box oriented, and thus scales to the complexity of modern IT systems, remains an unsolved problem.

6. *Failure prediction.* Several tools claim to possess failure prediction capabilities. They all belong to the category of either electrical or mechatronics (electro-mechanical) parts, where failure prediction is based on the historical and statistical data available in predefined libraries or the aging process is defined by the user. There are no tools (to our knowledge) that enable dynamic failure prediction for a running, distributed SOA system.

7. *Technical knowledge required.* Almost all quantitative tools require specialized knowledge in the area of fault-tolerant systems and formal modeling methods, such as Petri nets, Markov chains, or fault trees. This limits the applicability of surveyed tools, as the learning curve is often too steep to be practical. Consequently, the industry penetration is low. The notable exception is Figaro tool which offers general purpose, object oriented description language which is then transparently transformed into appropriate model-based notation.

8. *Monitoring and data gathering.* Tools from the benchmarking group, as well as tools from the process management group, offer instruments for system monitoring and data gathering. The methods are general and not availability assessment-specific. A better solution would be to develop an automated method for variable selection, to achieve parameterized system analysis.

9. *SOA Specifics.* General purpose modeling tools are primarily used in the design phase and follow the standard paradigm of modeling, evaluating, building (the system) and forgetting the model. In SOA, however, services and infrastructures are dynamic, and manually built models do not scale, as each change of the physical reality requires (manual) customization of the model. Taking into account shortage of availability modeling experts in modern enterprises, this procedure is simply too expensive and inefficient.

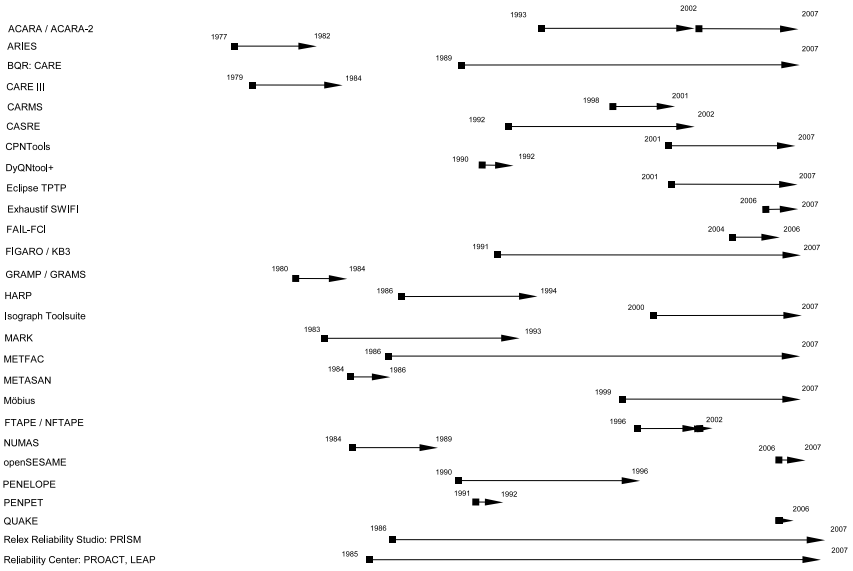

Figure 4.2: Historical development of availability assessment tools (part 1)

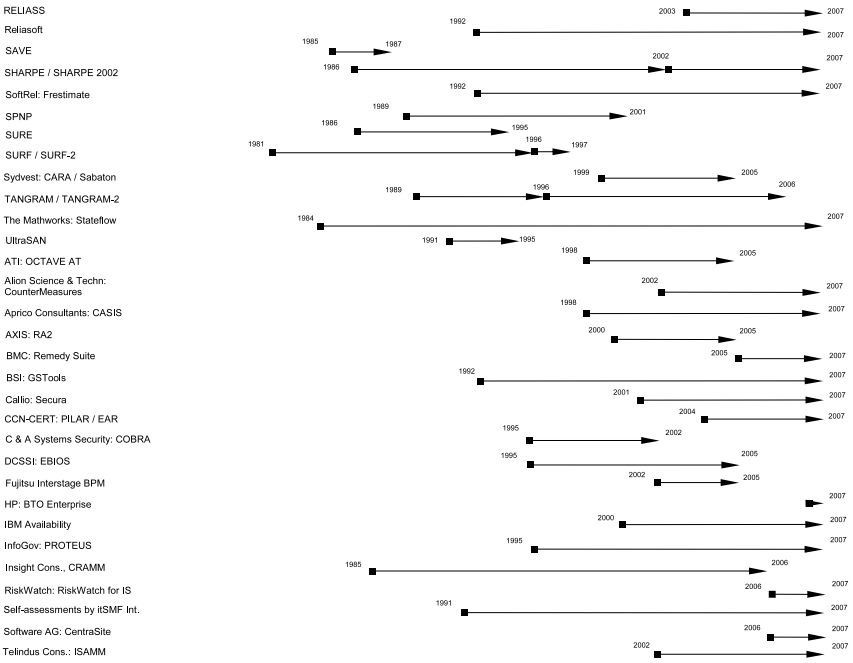

Figure 4.3: Historical development of availability assessment tools (part 2)

Name	License	Class	Model type	Platform	General purpose	Use cases
ACARA	Open-Channel-Foundation	Qualitative and analytical, IT-level	Statistical Monte-Carlo methods, Weibull and exponential distribution, RBD, models for early, random and wearing failures	Windows 98 or newer	Analysis of availability, lifecycle-costs and resource planing for the periodic reparable systems	NASA
Aprico ClearPriority	Commercial license	Quantitative, IT-level	Correlation-Rules, Data transformation	N/A	The tools collects log files for diverse systems, correlates the data and generates security alarms which are based on the predefined rules.	N/A
ARIES	N/A	Analytical, IT-level	Homogeneous Markov models, solved using Lagrange-Sylvester interpolation	APL and C implementation	Availability and lifecycle analysis of fault-tolerant systems	Teaching support
Axis: RA2	Commercial license	Qualitative, IT- and process-level	Risk analysis and assessment following ISO/IEC 17799 and 27001	Windows	Risk analysis following ISO/IEC 17799 and 27001. For each step of the process, the tool evaluates the risk and generates report.	Small and medium consulting companies in Asia (Singapur, Japan). No futher information available.
BQR Care	Academic and commercial license	Analytic, simulation, mechatronic	Failure Mode, Effects, and Criticality Analysis, fault trees, reliability block diagrams	Windows 2000, XP	Availability analysis of technical, electronic and mechanical (mechatronic) systems	BQR supports companies in the following sectors: air and space travel, automotive, chemical, (consumer) electronic, defense, military; oil, medicine, public transportation.
CALLIO Secura	Commercial and academic license, demo	Qualitative, process- IT-level	PDCA (Plan-Do-Check-Act) Model, Questionnaire	Windows and MS Access	The tool supports introduction of BS 7799 and ISO 17799 standards. It also supports development and management of Information-Security-Management-Systems (ISMS).	Concrete use cases not available, the customers are from the IT-consulting and e-Business sectors.
Cara-FaultTree	Commercial license, test version with limited functionality	Analytical, IT-level	Fault tree analysis	Windows 9x, NT, 2000	The tool specializes in the fault tree analysis only, limiting its general applicability.	Asset and risk management in the sectors of chemical, energy, transportation and oil industry.
CARE III	N/A	Analytical, IT-level	Fault trees, non-homogeneous Markov chains and semi-Markov chains	VAX	Availability assessment for very large, highly fault-tolerant systems	Aerospace industry
CARMS	Free	Analytical, IT-level	Markov chains	Windows 3.11 and later	Integrated tool for specification and simulation of Markov chains, specificaly tailored for time-dependent and prediction-oriented problems.	N/A
CASRE/ SMERFS	Open Channel Software	Quantitative/analytical, IT-level	Jelinski-Moranda Model, non-homogeneous Poisson process model	Windows, Unix	Measurements and assessment of software reliability. CASRE gives graphical representation of failure data, filtering and predicitions for a given software module.	Software reliability prediction.
COBRA	Commercial license, test version	Qualitative, process-level	Knowledge-based risk analysis	N/A	Risk assessment in security area. The relative importance of all threats and weak spots is identified and appropriate solutions and suggestions are generated.	N/A
Counter Measures	Commercial license (enterprise, standard, web survey), evaluation license	Qualitative, process-level	Questionnaire, customer-specific checklists, inspectors. Used for performing cost-benefit analysis	Windows 98/ ME /NT / 2000/ XP, Microsoft Office 2000 or newer	Risk management using US-NIST-800 and Circular-A-130 regulations, particulary for the IT-system, building and operation risk.	Physical security: motorways, buildings, critical infrastructure (oil).

Name	License	Class	Model type	Platform	General purpose	Use cases
CPNTOOLS	Commercial, military-government, academic	Analytical, IT-level	Colored Petri nets	Windows 2000, XP, Linux, Fedora Core 2	Specification, simulation and analysis of colored Petri nets. Support for distributed processes that communicate with each other and need synchronization.	Network protocols, software, workflows, and business processes; real-time embedded systems, air traffic control.
CRAMM	Commercial license, 30-day evaluation	Qualitative, process-level	Risk analysis and assessment following CRAMM (CCTA Risk Assessment and Management Method).	Windows	Used for automatization of the CRAMM process. CRAMM comprises of several steps that capture technical (IT, hardware, software) and non-technical aspects: identification and assessment of activities, analysis of threats and damages, choice of recommendations and countermeasures.	CRAMM is used by BAE Systems, IBM, General Motors, Royal Air Force, Swiss Bank, T-Mobile.
DyQNtool+	Academic license	Quantitative/ analytical, IT-level	Markov reward models	SUN-4	The tool applies the concept of dynamic queuing networks to assess reliability and performability using a formal description language.	N/A
EBIOS	Open source	Qualitative, process-level	EBIOS (Expression of Needs and Identification of Security Objectives) Method	Windows, Linux, Solaris	Supports the EBIOS method, and generates risk analysis and management steps for the five EBIOS phases.	Public sector, consulting companies.
Eclipse TPTP	Open source	Quantitative/ analytical, benchmarking and test, IT-level	PTP monitoring tools: analysis and filtering of log file data, pattern matching rules.	All major platforms and operating systems, for which JVM exists.	TPTP is a standardized, generic and expandable (meta) tool platform, for development of custom application in the area of testing and benchmarking. TPTP offers modules that allow for basic performance and test task generation.	Software, supports test description language TTCN-3 for communication-based applications.
Exhaustif	Commercial license	Quantitative, benchmarking, test; IT-level	Software Implemented Fault Injection (SWIFI)	RTEMS/ ERC32 and RTEMS/ Intel. Planned: Windows/ Intel, Linux/ Intel.	Execution of black-box and grey-box test based on the SWIFI method. The tool is used to improve reliability and availability properties of complex software systems during design time.	Beta-version is applied/ tested by EADS-Astrium.
FAIL-FCI	N/A	Quantitative-benchmarking and test, IT-level	Software-based procedures for fault injection.	Linux	Reliability estimation of cluster and grid systems. FAIL-FCI enables modeling and implementation of fault scenarios using an abstract fault description language.	Xtrem Web (P2P)
FIGARO/ KB3 Workbench	Commerical license, test version	Analytical, simulation, IT-level	Markov chains, Boolean logic driven Markov Process (BDMP), fault trees, reliability block diagrams, Petri nets.	Unix/ XWindow, MS Windows	Calculation of reliability, availability and performance using different models. Abstract systems can be modelled using knowledge database. Compilers and translators automatically generate appropriate formal models.	Reliability and security assessment of critical industrial processes (nuclear powerplants, chemical and oil industry).
Fujitsu Interstage	Commercial license, test version	Qualitative, service-level	Workflow (BPMN, BPEL, XPDL, Wf-XML)	Windows 2000, 2003, XP, Solaris 9, Red Hat Linux, ES 4.0, HP-UX, IBM AIX 5.3	The tool comprises a workflow-based business process manager that covers the following lifecycle: process integration, automatization, modelling, management and optimization. The last three steps allows for treatment and assessment of availability indicators.	Banks und financial institutions, energy sector, travel and tourism, enterprise software systems.
GRAMP/ GRAMS	N/A	Simulation, IT-level	Continuous-time Markov models (GRAMP), solved using discrete Monte-Carlo simulations (GRAMS)	Fotran 77, VAX implementation	GRAMP: Fixed charge for repair, Failure charge for breakdown, Mission length, Acquisition cost (system level and module level), Average repair cost, Reliability requirement, Opportunistic maintenance option. GRAMS: Assessment of reliability and lifecycle costs.	Full-Authority Fault-Tolerant Electronic Engine Control (FAFTEEC) Program

Name	License	Class	Model type	Platform	General purpose	Use cases
GSTOOL	Free for government use	Qualitative, process- and IT-level	Risk analysis following BSI-Grundschutz recommendation/ standard.	Windows	Assistance in design and implementation of BSI-Standard 100-3.	Management of IT-infrastructure and processes in the public (government) sector.
HARP	N/A	Analytical, IT-level	Fault Occurence and Repair Model (FORM), Fault/ Error Handling Model (FEHM), Markov chains	MS DOS, Windows, OS/2, DEC VMS and Ultrix, Berkeley Unix 4.2, AT&T Unix 6.2	Assessment of reliability and availability using fault trees as input.	N/A
IBM High Availability Services	N/A	Qualitative, service-level	Questionnaire	N/A	IBM High Availability Services is a consulting service supporting requirement planning, design, construction, implementation, and management of complex infrastructures, with the goal of avoiding expensive breakdowns. It also offers service availability assessment, as part of SLA parametrization.	N/A
IBM Tivoli Availability Process Manager	Commercial license, academic and test versions	Qualitative, IT- and process-level	Component Failure Impact Analysis (CFIA) and Availability management, as specified by ITIL.	IBM AIX 5.2, 5.3, RedHat Linux, Microsoft Windows	Tivoli Availability Process Manager enables availability assessment and indident priority assignment according to their impact not only on IT infrastructure, but also on critical business processes. In that way, the tool establishes a connection between IT and business layers. It offers availability management (incident management) based on ITIL and determines business impact of a failure.	Automotive, financial, IT-service, energy, administration, healt and media sectors.
ISAMM	N/A	Qualitative, process-level	Risk analysis and assessment following the Information Security and Monitoring Method (ISAMM)	N/A	The tool is used for risk management, and calculates the optimal security repair plan based on the Return on Security Investment (ROSI) values.	Academic (security analysis at the University of Kaiserslautern)
IsoGraph AttackTree+	Commercial license, test version	Analytical, simulation, service-level	Attack tree	MS Windows	Attack tree enables precise modelling of threats to the system security using a graphical language.	Internet applications, bank applications, networks.
IsoGraph AvSim+	Commercial license, test version	Simulation, IT-level	Fault trees, Network (reliability-block) diagrams	MS Windows	Availability and reliability simulation of complex, dependent systems.	Air and space industry, aefense, automotive, railway, chemical, oil, health.
IsoGraph FaultTree+	Commercial license, test version	Analytical, IT-level	Fault and event trees, Markov models	MS Windows	Availability analysis using fault and event trees. Customised Markov models may be linked to event in the fault or event tree diagram. Independent analysis of fault trees, event trees and Markov models is also possible.	Air and space industry, aefense, automotive, railway, chemical, oil, health
IsoGraph NAP	Commercial license, test version	Analytical, IT-level	Extended reliability block diagram	MS WIndows	Prediction of availability and reliability of communication networks.	Communication networks.
IsoGraph Reliability Workbench	Commercial license, test version	Analytical and simulation, IT-level	Various prediction models, hierarchical block diagrams	MS Windows	Reliability Workbench is an integrated environment for performing Reliability Prediction, Maintainability Prediction, Failure Mode Effect and Criticality Analysis (FMECA), Reliability Block Diagram (RBD) analysis, Reliability Allocation, Fault Tree Analysis, Event Tree Analysis and Markov Analysis.	Air and space industry, aefense, automotive, railway, chemical, oil, health
ItSMF	Free	Qualitative, process-level	Questionnaire	Online/ MS Excel	ITIL best practice evaluation.	IT-consultants, government

Name	License	Class	Model type	Platform	General purpose	Use cases
MARK 1	N/A	Analytical, IT-level	Discrete-state continuous-time Markov model	PL/1	Evaluation of reliability of complex systems whose characteristics can be modelled using Markov chains. Calculates state probabilities as functions of time, MTBF and average state occupancy probability.	Reliability prediction for the fault-tolerant multiprocessor (FTMP) developed for NASA. Airplane systems and safing systems in nuclear reactors.
Mathworks Stateflow	Commercial license, academic license, time-limited demo version	Quantitativ-simulation, IT-level	State charts, finite state machines, temporal logic	Windows 2000, XP, Vista, Linux, Solaris 8.x, MacOS on Intel	Simulation tool for the event based systems. It extends state charts with the following concepts: control flow, truth tables, temporal operators, event-based broadcast.	Energy and power sector.
Mercury BTO Enterprise Solutions	Commercial license, test version	Qualitative, IT- and process-level	ITIL service management, COBIT quality standards	Windows, Linux/ Unix	Mercury BTO (Business Technology Optimization) Enterprise is a suite of applications (tools) that supports implementation of ITIL service management. They are technologically bound by dashboards, a CMDB (Configuration Management DataBase), and a workflow engine that automates and integrates service management processes. Mercury Project and Portfolio Management Center provides CobiT support (it also provides support for other quality programs).	N/A
METASAN	N/A	Analytical and simulation, IT-level	Stohastic activity networks (SAN)	Unix	Evaluation of performance and reliability.	N/A
METFAC	Commercial license, academic license, evaluation license	Analytical and simulation, IT-level	Finite continuous-time Markov chain models with reward rates associated with their states.	Linux kernel 2.4.4 and above; Solaris 2.x; and, on demand, any other UNIX variant with C- shell and an ANSI/ISO (standard 89/90) C compiler	Analysis of performance, dependability and performability of complex systems through rewarded continuous-time Markov chain models.	A Reliability Model of a 5-level RAID Storage Subsystem, Reliability Model of a Storage System, Performability Model of a Multiprocessor System, Grid cluster computing systems, Storage area networks, Communication networks
Möbius	Commercial license, academic and test versions	Quantitative (analytical and simulation), IT-level	Stochastic activity networks (SAN), Queuing systems, stochastic Petri nets, stochastic process algebra, fault trees, combinatorial block diagrams.	Windows 2000, XP, Fedora Core 3 and newer	modeling the behavior of complex systems. Originally developed for studying the reliability, availability, and performance of computer and network systems, it can be used for a broad range of discrete-event systems, from biochemical reactions within genes to the effects of malicious attackers on secure computer systems, in addition to the original applications.	IT-systems and networks, telecommunication software and hardware systems, aerospace and aeronautical, commercial and government systems and networks, biological systems.
NFTAPE	Academic license	Quantitative benchmarking and test, IT-level	Software-based fault injection	Solaris, Linux	NFTAPE is a software-based environment for automatical assessment of network reliability using fault injection methods.	Motorola IDEN MicroListe (base station controller), DHCP protocol, SSH and FTP services.
NUMAS	N/A	Analytical, IT-level	Queuing networks, Markov chains	Sun	Performance analysis tool, enabling expansion of node properties with fault-tolerance parameters.	N/A

Name	License	Class	Model type	Platform	General purpose	Use cases
OCTAVE	Commercial license	Qualitative, IT- and process-level	Risk analysis	N/A	Developed by the Advanced Technology Institute to support the Operationally Critical Threats, Assets and Vulnerability Evaluation (OCTAVE) methodology. It assists in the data collection phase, data organization and report generation.	Health (OCTAVE-HIPAA, OCTAVE-JCAHO)
OpenSesame	Academical license, commercial available upon request	Quantiative, analytical/ simulation, IT-level	Reliability block diagram, Fault tree, Stochastic Petri nets	Linux/ Unix	Combines user friendly techniques of combinatorial methods (RBD/FT) with expression power of state-based modeling methods (Petri nets). Component dependence can also be specified.	(Web)servers, Telco networks, energy sector.
PENELOPE	N/A	Qualitative and quantitative, IT-level	Extended Markov reward models, controlled stochastic Petri nets	Sun-4	Performance analysis and optimisation	Emergency supply model
PENPET	N/A	Quantitative - analytical, qualitative, IT-level	Generalized stochastic Petri nets, Markov reward models	C implementation for Unix	Analysis of performance and reliability of fault-tolerant systems.	Multiprocessor systems.
PILAR/ EAR	PILAR is licensed for Spain government use only, EAR is available in commercial test and versions.	Qualitative and quantitive, process-level	Methodology for Information Systems Risk Analysis and Management (MAGERIT), attack trees, process descriptions, audits; Uses data flow diagrams, process charts, boolean functions.	Windows/ Linux	PILAR implements and extends the MAGERIT-process developed by the Spanish Defense Ministry. It comprises various qualitative and quantitative risk analysis methods, as well as risk management, business-impact-analysis and continuity-of-operations.	PILAR: Spanish adminstration.
PROTEUS	Commecrial license, Web demonstration	Qualitative, and process-IT-level	Risk analysis and assessment	Client: MS Windows NT/ 2000/ XP; server: MS Windows Server 2003, Linux	Webserver-based supporting information security and risk management standards (ISO/ IEC 17799 and BS ISO / IEC 27001, BS 25999, Cobit, PCI DSS...)	N/A
QUAKE	N/A	Quantitative - benchmarking and test, IT-level	Fault injection, time series analysis, benchmark	N/A	Reliability assessment of grids and Web services	Grids and Web services, no additional details.
Relex Reliability Studio, PRISM	Commercial license, evaluation version	Quantitative - analytical/ simulation, qualitative, IT- and process-level	Markov models, reliability block diagrams with Monte-Carlo simulations, FMEA/FMECA, FTA/ETA, FRACAS, Human-Factor-Risk-Analysis	MS Windows	Assessment of reliability, performance and maintainability, supporting methods for determining fault criticality and availability modeling.	Air and space industry, automotive, oil, health and medicine technique, telecommunications.
Reliability Center: Proact, LEAP	Commercial license	Quantitative - analytical and simulation, qualitative, IT-level	Root-cause-analysis, FMEA and opportunity analysis	MS Windows	Supports the Proact process model, a variant of root-cause analysis method. LEAP implements FMEA and opportunity analysis.	Air and space (NASA), railway (Amtrak), food (Baccardi), automotive (General Motors), multi-concerns (General Electric), power (Virginia Powers), oil (Shell).
Reliasoft	Commercial and evaluation license	Quantitative - analytical, qualitative, IT-level	Life data analysis, RCM, FMEA/ FMECA, RCM, RBD, FRACAS, stochastic event simulation	MS Windows	Software availability analysis	System and product manufacturing, environment planing.
RELIASS	Commercial license, evaluation version	Quantitative - analytical and simulation, IT-level	FMECA, RCM, corrective and preventive maintenance, sensitivity analysis (ASENT), RBD, Monte-Carlo simulations, Markov chains, Fault trees, Event analysis, Weibull distribution, reliability predictions, Weak-link analysis and phased simulation.	Windows 98 and newer	Reliability, availability, performance and maintenance assessment.	Lockheed Martin (JSF project)

Name	License	Class	Model type	Platform	General purpose	Use cases
RemedySuite	Commercial license	Qualitative, process-level	Remedy Software for IT-Service-Management (ITSM) model	Windows, Unix/ Linux	Improvement of project management quality through automatization of ITIL-processess.	TeliaSonera, Infineon, Vodafone, Agfa, Dell.
Risk Watch	Commercial license	Qualitative, IT- and process-level	Risk analysis and assessment	MS Windows/ Office	Automatic risk analysis based on the customer-specific knowledge base.	N/A
Sabaton	Commercial license, test version with limited functionality	Analytical, IT-level	FMEA/CA	MS Windows	Supports FMEA (Failure Mode and Effects Analysis) and FMECA (Failure Mode, Effects and Criticality Analysis), that are used to cover possible failures and failure types during the product and system development.	Air and space, telecommunications, transaction systems.
SAVE	N/A	Analytical and simulation, IT-level	Homogeneous Markov chains	Fortran 77	Solving probabilistic models of system availability and reliability of mission-oriented and continuously operating systems.	Space computers and avionics (non-repairable systems) and telephone switching systems, general purpose computer systems, transaction processing systems (repairable systems).
SHARPE 2002	Academic license, commercial license	Quantitative, analytical and simulation, IT-level	Markov chains, semi-Markov chains, reliability block diagrams, fault trees, reliability graphs, queuing networks, generalized stochastic Petri nets, serial-parallel graphs.	Windows, Linux, Solaris, JVM	The tools offers specification languages and solvers for most model types used for availability, reliability and performance modeling.	More than 280 commercial installations, specifics not given.
SoftRel LLC: Frestimate	Standard/ manager edition, Metrics package	Analytical, software	Customized models based on correlation	MS Windows	Software reliability assessment with respect to the expected number of faults per 1000 lines of code.	Defense, Air and space industry.
Software AG Centrasite	Commercial license, test version	Qualitative, service-level	Metadata repository supporting policy, change control, maintenance, automation, dependency analysis and reporting at the business process (service) level.	Platform-independent	Centrasite is platform for Service Oriented Architecture (SOA) Governance. SOA governance covers two aspects: 1) service implementation and 2) IT governance at the business process level. Governance priorities are performance, risk management, service availability and aligning IT infrastructure with business goals.	Fujitsu, Novell, Software AG,Belgian National Railway Company, Commerzbank, DaimlerChrysler, Nissan Motors, Scandinvian Airlines, Vodafone, Volkswagen, ZDF
SPNP	Academic and commercial license	Quantitative, IT-level	Stochastic reward net (SRN), fluid stochastic Petri net (FSPN)	MS-DOS, Solaris, Linux	Performance, reliability, availability analysis of complex systems.	Database systems, ATM-networks.
SURE	N/A	Analytical, IT-level	Semi-Markov chains	Solaris, Linux, MS Windows 98	Reliability analysis of fault-tolerant architectures, calculates upper and lower bounds of failure probability.	N/A
SURF-2	Commercial license, academic license	Analytical and simulation, IT-level	Markov chains, generalized stochastic Petri nets	SUN OS 4.1.x or Solaris 2.x	Hardware and software reliability estimation.	N/A
TANGRAM	N/A	Depends on the application domain	Depends on the application domain	Sun-3	Framework which enables plugging of different models, their evaluation and reporting.	N/A

Table 4.1: Overview of the surveyed availability assessment tools

Chapter 5

Service and Process Availability

As discussed in Chapter 2, business processes and services are based on the ICT-layer components, their availability therefore depends directly on the availability properties of the underlying ICT-components. In the previous chapter we investigated limitations of existing reliability and availability modeling tools when applied to complex business processes and service-oriented applications. To overcome these limitations we introduce a mapping process that describes dependencies between the ICT-layer, service and business process level. Effectively, process and service availability is determined, analytically or through simulation, as a function of the ICT-layer components' availability, where underlying availability models are generated automatically.

Figure 5.1 sketches the proposed service availability assessment process. Based on the service or process description and the infrastructure data that is gathered, steps of service or process execution are mapped to infrastructure elements on which they depend. Based on this mapping, availability model (reliability block diagram or Markov chain) is automatically derived and parameterized. The model thus obtained is then solved to provide metrics such as steady-state, interval, instantaneous or user-perceived availability. All steps are automated except the initial process/service description.

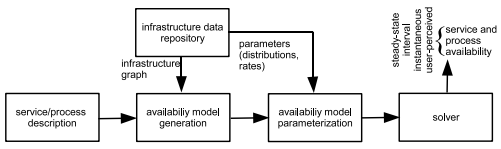

Figure 5.1: Proposed availability assessment steps

The mapping and availability assessment process sketched above comprises following steps:

1. Business process is identified and described using a process modeling language, such as BPMN (in this chapter) or UML activity diagram. Optionally, the required (target) process availability is defined.

2. For each business process activity, services enacting it are iteratively described using the same process modeling formalism, until all activities are represented as compositions of atomic services. Similarly to step 1, required availability of each service can be also defined.

3. Infrastructure data is collected by a network management system that may be integrated in a larger configuration management database (CMDB) system. In this step, communication and computation topology is extracted in form of the infrastructure graph. Information about supporting infrastructure, such as power supply or air-conditioning devices, are either taken from specialized tools (e.g., GSTool) or added manually to the infrastructure graph.

4. Services are iteratively mapped to infrastructure elements contained in the infrastructure graph, in a process that is inversion of step 2: atomic services are mapped first, followed by composite services. Each step of atomic or composite service execution has source S and destination D in the graph. The task is to find all paths between S and D. The paths are then transformed into Boolean expressions by applying operator AND between nodes that belong to the same path. If more than one path between source and destination exist, operator OR is applied to the expressions defining single paths. If two nodes are executing concurrently in the service description, they are serialized using operator AND, however if only one or k-out-of-n have to be executed, they are joined with operator OR or expanded to combination of AND-OR operators (in k-out-of-n case). The resulting expressions are then minimized. The Boolean equations thus minimized represent communication paths.

5. Step 4 is repeated for the business process. Effectively, Boolean equations are derived that express functional dependency between business process, service and ICT-layer elements, analogously to service-ICT mapping from step 4.

6. The expressions obtained in steps 4 and 5 are transformed to appropriate model for availability assessment. In the following sections it is demonstrated how to transform Boolean expression into reliability block diagrams (RBD)/fault trees (FT) and Markov chains.

7. Availability of the business process, composite service or atomic service is calculated by solving/simulating the model generated in the previous step, using an existing solver. Provided availability of each service and business process can be compared with required availability, if one was defined in steps 1-2.

Steps 4 and 5 require identification of all paths between two nodes in a graph. In the worst case of a complete connectivity graph, the space/time complexity of a recursive algorithm reaches prohibitive $O(n!)$. However, real intranets are mostly tree structures, where loops may be created only by routers

whose number in a network is limited. Networks are designed such that a moderate number of switches are connected to the routers and numerous hosts are then connected to the switches, creating a tree (see infrastructure graph examples in the following sections). Such sparse structure limits the algorithm complexity, and it remains polynomial (for a tree it is linear as there is only one path for each pair of nodes). First two steps are human-dependant but they are performed only once per process/service, when it is added to the enterprise or when its definitions is changed. Furthermore, process and service modeling can be supported using template-based mechanisms and service description repositories. Steps three through seven can be automated and they can autonomously adapt to changes: e.g., a change in the infrastructure triggers the update of a CMDB which initiates new availability assessment.

We will now cover algorithm steps in more detail, explaining how to perform infrastructure graph generation (step 3), how to define business process/service mappings (step 4), how to define communication paths (steps 4 and 5) and finally how to generate the appropriate availability model (step 6). Results presented in the following sections are partially based on our previous work from [147] and [137].

5.1 Mapping BPMN activities

For each deployed (observed) instance of a BPMN model, deployment descriptor is generated, similar in structure and purpose to BPEL deployment descriptors. Each activity in a BPMN model is tagged with `<partnerLink>` element, which can either specify a composite service, atomic service or ICT-layer element (or several of them) which implement this activity. The only constraint is that atomic service activities must reference ICT-level elements. In that way, hierarchical tagging of processes/services is performed. A deployment descriptor reads the activities from an abstract BPMN model (persisted in XMI format) and generates a set of `<provide>` and `<invoke>` elements for each `<partnerLink>` element. Every `<partnerLink>` used with `<receive>` activity must be matched with `<provide>` element, and every `<partnerLink>` used in `<invoke>` activity must be matched with `<invoke>` element in deployment descriptor. The root element, `<deploy>`, contains a list of all deployed processes, that is, single deployment descriptor can be used for more than one process (BPMN) model:

```
<deploy>
 <process ...>*
 { other elements }
 </process>
</deploy>
```

Each process is identified by its qualified name and specifies bindings for provided and invoked composite/atomic services or ICT-layer elements:

```
<process name = QName  fileName = String?  implName = String? >
 (<provide> | <invoke>)*
 { other elements }
</process>
```

Each process element must provide a name attribute with the qualified name of the BPMN process. This is parsed from XMI file. Optionally, the fileName attribute can be used to specify location of the BPMN process definition. The implName attribute is used to specify an endpoint which implements particular partnerLink. Each `<process>` element must further enumerate services or components provided by the process and bind each service or component to an endpoint. This is done through `<provide>` elements which associate partnerLink with endpoint. Essentially, in this step it is determined which components implement certain BPMN activity and this information is stored in an endpoint:

```
<provide partnerLink=NCName>
  <service name = QName port = pName?>*
</provide>
```

However, whereas in the BPEL deployment descriptor a partnerLink can be implemented only with a WSDL endpoint, we do not impose this restriction. The partnerLink can be implemented by any class from Figure 2.8 derived from the class `ICTLayerComponent`. Some of the required information can be extracted manually and associated with elements of the infrastructure graph, for example if a software component is deployed in an application server, this information can be extracted from the component deployment descriptor (usually `web.xml` file). In some cases manual deployment descriptor generation is required (see the Publishing Case Study, where manual activities of editors must be defined by hand in the mapping process). Let us observe a business process where client initiates the process with an input which is forwarded to `serviceA`, it computes the result which is asynchronously sent in parallel to `serviceB` and `serviceC`, and finally, a selection between two computed values is performed. The following is XML (exported BPEL) representation of such a business process:

```
<process name="test">
  <partnerLinks>
    <partnerLink name="client"/>
    <partnerLink name="serviceA"/>
    <partnerLink name="serviceB"/>
    <partnerLink name="serviceC"/>
  </partnerLinks>
  <variables>
    <variable name="procesInput"/>
    <variable name="AInput"/>
    <variable name="AOutput"/>
    <variable name="BCInput"/>
    <variable name="BOutput"/>
    <variable name="COutput"/>
    <variable name="processOutput"/>
    <variable name="AError"/>
  </variables>
  <sequence>
    <receive name="client" variable="processInput"/>
    <assign><copy><from variable="processInput"/>
    <to variable="AInput"/></copy></assign>
   <scope>
     <faultHandlers>
       <catch faultName="faultA" faultVariable="AError"/>
     </faultHandlers>
     <sequence>
```

```
      <invoke name="invokeA" partnerLink="serviceA"‘
      inputVariable="AInput" outputVariable="AOutput"/>
    </sequence>
  </scope>
  <assign><copy><from variable="AOutput"/><to variable="BCInput"/>
  </copy></assign>
  <flow>
    <sequence>
      <invoke name="invokeB" partnerLink="serviceB"
      inputVariable="BCInput"/>
      <receive name="receive_invokeB"partnerLink="serviceB"
      variable="BOutput"/>
    </sequence>
    <sequence>
      <invoke name="invokeC" partnerLink="serviceC"
      inputVariable="BCInput"/>
      <receive name="receive_invokeC" partnerLink="serviceC" variable="COutput"/>
    </sequence>
  </flow>
  <switch><case>
    <!-- assign value to processOutput -->
    </case></switch>
    <invoke name="reply" partnerLink="client" inputVariable="processOutput"/>
  </sequence>
</process>
```

This is a general business process model which can be deployed in numerous infrastructures. The task of the mapping procedure is to determine elements in the given infrastructure graph that implement (enable) particular process steps/activities. One possible deployment descriptor would have the following form:

```
<deploy><process name="test">
  <provide partnerLink="client">
    <service name="clientWebForm" port="foo:8080/client"/>
  </provide>
  <invoke partnerLink="serviceA">
    <service name="SAPService" port="BOR.IDOC.SEGMENT.abc/>
    <service name="BackupService" port="jdbc:storedProc"/>
  </invoke>
  <invoke partnerLink="serviceB">
    <service name="WS1" port="foo:serviceB.wsdl"/>
    <service name="auth" port="foo:auth.wsdl"/>
  </invoke>
  <invoke partnerLink="serviceC">
    <service name="EJB1" port="jndi:serviceC"/>
    <service name="auth" port="foo:auth.wsdl"/>
  </invoke>
  <invoke partnerLink="client">
    <service name="ClientFile" port="ssh user@host/>
    <service name="ClientFile" port="ftp user@host/>
  </invoke>
</process></deploy>
```

It can be seen that initial client activity is performed by the manual input using Web form, serviceA is implemented as the combination of SAP IDOC (structured file) call and JDBC backup stored procedure, serviceB and serviceC are implemented as Web service and Enterprise Java Bean respectively, and delivery of the results to the client is peformed as file transfer using ssh and ftp protocols. Note how client definitions differ in `<provide>` and `<invoke>` roles. Also, the component `auth` is used to invoke both `ServiceB` and `ServiceC`. This

shows that, although the process description specifies parallel execution of both services, internally (when deployed) they both depend on a single component, which represents a single point of failure. This information is neither obvious nor possible to include in the process/service description. Finally, note that deployment descriptor is not limited to WSDL endpoints only.

After service execuction steps have been mapped, all process/service activities are associated with the underlying infrastructure graph elements, this information is persisted in deployment descriptor, and the process of communication path generation can now take place.

5.2 Infrastructure graph generation

One of the prerequisites for successful availability modeling of services and business processes is the accurate information on characteristics of infrastructure components, their organization and mutual relationships. Services and business processes are deployed on the infrastructure layer that comprises technical infrastructure (hardware, network, supporting infrastructure), software and IT-personnel categories (Figure 2.8).

In modern enterprises, the number of components in the infrastructure and their diversity makes manual management of this information difficult, if not impossible. Network monitoring/configuration management database (CMDB) systems are used for easier and more accurate information collection (e.g., they may automatically detect node or network configuration changes). Monitoring tools are primarily used to track status of services and devices in the network. They may be configured to raise alarms if a device or service becomes unavailable, thus improving the time to repair. The data on service and device availability is preserved and can be used for MTTF and MTTR parameter estimation that is required for availability assessment.

Automatic network discovery is a desirable property of network monitoring systems, but it is not supported by all of them: for instance, a popular open source monitoring tool Nagios requires tedious manual configuration process in which a user defines monitored devices and services. The process is lengthy, inflexible in presence of changes and it does not provide network's Layer 2 topology.

CMDB systems are considerably more powerful, enterprise-ready solutions (e.g., IBM Tivoli, HP OpenView, Fujitsu Interstage). They are capable of automatic device, application and service discovery. They additionally manage configurations and their histories, allowing expedite problem resolution. Compared with open source solutions, they have wider set of tools at their disposal, such as the agent-less scanning of hosts in the network and built-in support for numerous vendor-dependent protocols and applications. Agent-less configuration and performance management is performed from a centralized server that locally runs software which accesses nodes in the network and gathers predefined data. For successful agent-less scanning, it is necessary that data collection server has credentials of managed devices and that a managed device supports some form of remote access (e.g., SSH). Agent-less collection is performed in

part through SNMP protocol [7], but in order to gather data about the application configuration, specialized software must be deployed at the server. Custom plug-ins considerably improve the detail level of collected data. For instance, HP Discovery and Dependency Mapping [121] is capable of gathering application specific information, such as the configuration of a WebSphere server or a tablespace configuration in Oracle.

Agent-based approach, on the other hand, requires installation of specific software agents on target nodes, but in addition to status monitoring, they frequently provide node and application management. For instance, IBM Tivoli Monitoring provides custom agents for SAP solutions that monitor the system and detect predefined exception conditions (e.g., exceeding a performance threshold), investigate the exception causes and schedule work/automate some manual tasks.

Collecting configuration of nodes and services deployed in the network is important for the later step in the availability assessment process: construction of communication paths. For example, a large organization hosts a set of mail servers. Each of the mail clients is associated with one of them. It is not possible to say which server is responsible for the assessed client and to evaluate the impact of network availability on their communication, based only on the general BPMN description of the email service. It is the task of the configuration collection modules (agent or agent-less) to provide the actual information that is used to instantiate the abstract BPMN service description with accurate data. If a configuration collection system does not exist in the organization, it is the user's task to specify this data.

As an example of a network monitoring system, we will briefly describe OpenNMS, which is an open source network monitoring system that we use in our prototypical implementation (see Section 5.8), capable of automatic node, topology and service discovery. It employs IP-range scan to discover devices and then performs port scan of discovered nodes to determine services activated at them. Discovery rate can be altered, in order to avoid network overloading. The data is preserved in PostgreSQL database for later use by OpenNMS or other applications.

Figure 5.2 shows a part of the database structure that is used by OpenNMS. Table `node` holds the basic information about nodes, such as their identification, operating system and domain name. Each node has one or more network interfaces (table `atinterface`) and belongs to a LAN network (table `vlan`). Different types of LAN networks are supported, such as Ethernet, token-ring, FDDI. Services are deployed on nodes (tables `ifservices` and `service`). The service status is periodically polled. Data about node and service outages and restorations are preserved in form of events.

OpenNMS is also capable of layer 2 topology discovery, although it is not enabled by default. The topology discovery requires activation and configuration of additional modules. In order to function properly, the topology discovery process requires that network devices (bridges, routers) support SNMP protocol. It discovers routers (data about their interfaces is saved into table `iprouteinterface`) and bridges (table `stpnode`). Table `datalinkinterface` stores data about network structure and inter-node connections.

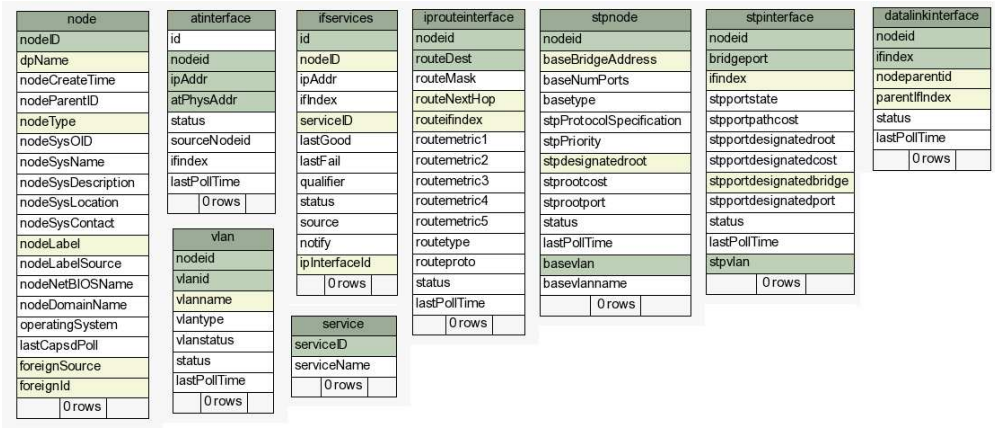

Figure 5.2: Partial OpenNMS database schema

It is obvious that discovery of data about the technical infrastructure and people working in an enterprise is not fully automatable. However, there are other tools which are used for collection and management of this data. If such tools are employed in the enterprise, their data can be also used to improve accuracy of the proposed availability assessment process. We will illustrate this on the example of GSTool [6], which primarily supports users in preparing, administrating and updating IT security concepts that meet the IT Baseline Protection Manual defined by the German Federal Office for Information Security (BSI). In addition to security-related information, the tool manages information about technical infrastructure and personnel that is highly relevant for the availability assessment.

Figure 5.3 shows how the tool captures information about buildings, rooms, IT systems and employees. In comparison with network monitoring tools, the level of details about IT systems is considerably reduced, but on the other side we can now clearly see that the Linux server is placed in the server room, which has a specialized administrator associated with it. Server room is located in Building 1 of the organization. A secretary is working in the ordinary office space within the same building and uses the client computer with Windows XP operating system and Outlook mail client.

The tool can collect additional data on technical infrastructure elements. As we can see in Figure 5.4, server rooms in Building 1, where our sample Linux server is located, have overvoltage protection, emergency circuit breakers, local UPS and redundant air-conditioning system that is connected to the independent UPS (indicated by the green color). On the other hand, a study of technical and organizational requirements for server rooms and the support for remote reporting of system malfunction do not exist (indicated by the red color). Finally, redundancy of the technical infrastructure is only partially implemented (indicated by the yellow color). Similar data may be defined at the organization level. Same as in the OpenNMS, data is preserved in a relational database and can be easily accessed from other applications, such the tool pro-

totype presented in section 5.8.

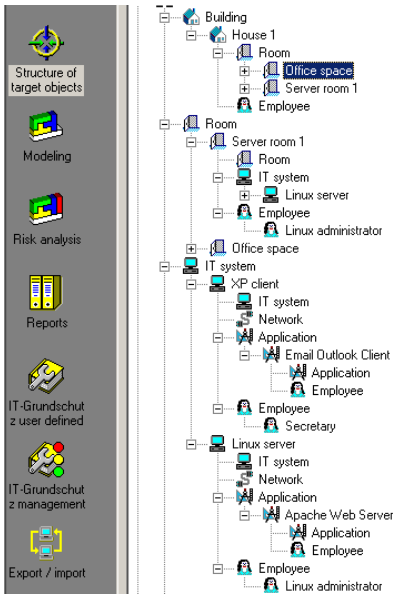

Figure 5.3: Hierarchical organization of ICT infrastructure elements in GSTool

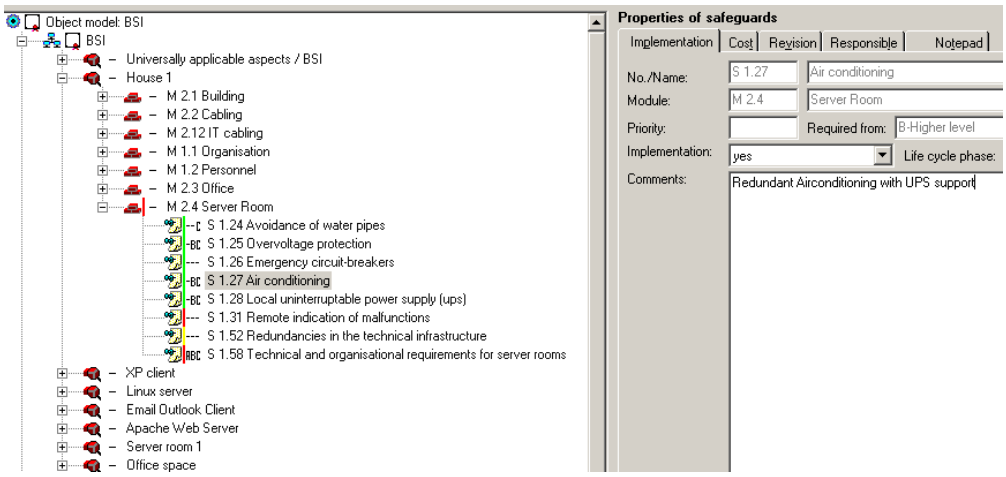

Figure 5.4: Technical infrastructure description in GSTool

Before we can proceed with the mapping process, the infrastructure graph must be created. Data for its creation originate from network and system management tools. The data must be collected and, if necessary, pre-processed before they can be used for availability assessment. As a formal model for the infrastructure layer we use annotated graphs. In addition to the connectivity that is captured in traditional graphs, annotated graphs enable association of attributes to nodes and edges in the graph. The attributes are used to classify nodes and attach additional properties to them. Some attributes are common

for all nodes and edges in the graph, such as their type and unique identification number. Nodes and edges may have additional attributes, organized in lists of (key, value) pairs. Graph nodes are divided in three classes:

- Network node. Represents a physical node such as router or server. For each active network interface, there exists a link to a node that holds the second endpoint of the connection. Mandatory attributes are MTTF and MTTR. Additional properties may be included as well in the list of annotations (e.g., presence of air-conditioning, UPS, physical security, etc.).

- Atomic service node. Usually these are software services associated with a network node. Atomic service node must also have the MTTR and MTTF attributes. There is additional Boolean attribute that clarifies whether the service's MTTR and MTTF properties supersede the properties of associated network node or they are independent and to be combined in evaluation (in practice this depends how the values have been estimated – as a joint characteristics of the network node and service, or independently). Unlike the network node, it does not include data on technical infrastructure.

- User/operator node. Operator is associated with one or more nodes of other types. Its mandatory parameter is the probability of operator errors [116] that influence availability of the node/service to which the operator is associated.

The node-service, operator-node and operator-service edges in the graph model do not have attrributes, while edges that model node connectivity are annotated with MTTF and MTTR values. If a network node is associated with a single service in the evaluation process, their representation may be joined, avoiding unnecessary complexity (this is the approach we have used in our case studies). The MTTF and MTTR parameters that are associated with the network node are used to model the joint behavior of nodes and software services.

Once the infrastructure graph is prepared, the BPMN description can be mapped to it. From the deployment descriptor `<provide>` and `<invoke>` fields are extracted. Appropriate service endpoints are located in the infrastructure graph (source S and destination D of the service invocation). Again, in ideal case, this step is performed automatically. In simpler scenarios, it is rather easy to implement it (e.g., agent-less scanning may be used to determine primary and secondary DNS of a client computer or server in the network) but models that are encountered in practice are considerably more complex, requiring human interaction. For instance, it cannot be resolved automatically which user is part of the business process if a set of users, each with a different skill, is associated with an atomic service that participates in a business process. Once the source and destination are known, an algorithm for detection of all paths between these two endpoints is applied. For each independent path that is discovered, a boolean expression is derived: all vertices on the path are included as well as

the annotated edges. Within a single path, terms are connected with the AND operator. If independent paths exist, they are joined with the OR operator. First minimization of the Boolean expressions may be executed at this point (examples how is this performed are found in the case studies).

During the path generation process, vertices that compose the paths are queried for needed characteristics using the key field. If it is not defined for a node, the user is prompted to specify the missing parameter. This is done in order to support partial specification in infrastructure modeling process, and to avoid unnecessary detail level: if the evaluated service uses only a small subsection of the complete infrastructure, there is no need that user predefines parameters in the whole infrastructure.

This step is iterated for all atomic activities in the BPMN diagram, until the complete set of Boolean expressions is obtained. They are then joined into a single Boolean expression using the rule that all execution steps (i.e., Boolean expressions that describe them) that are sequential in the flow are joined with AND operator and OR operator is used only for the unconditional flow branching. Finally, the obtained expression is minimized and forwareded to the next step – automatic generation of availability models.

5.3 Automatic Generation of Availability Models

Based on the minimized Boolean equations obtained in previous steps of the algorithm, availability models are generated. We currently support two types of availability models: combinatorial models (reliability block diagrams and fault-trees) and state-space models (Markov chains). Combinatorial models are very easy to generate, however, their expressive power is low and metrics that can be calculated is limited. State-space models are complicated to generate, but can cover broader class of systems, capture different behavior and offer more advanced transient and instantaneous assertion possibilities. Furthermore, combinatorial models cannot be generated for all types of business processes, e.g., when priorities or coverage are included (e.g., the Amazon EC-2 example from introduction). Here we give a very simple procedure to transform Boolean equations into an RBD (there exists a bijection between RBD and FT):

- Blocks of the RDB are all terms appearing in the minimized Boolean equation.
- If two terms are connected with operator &, RBD blocks are generated for both terms and placed into serial configuration.
- If two terms are connected with operator ||, RBD blocks are generated for both terms and placed into parallel configuration.
- The process is performed in a single pass, parsing the equation from left to right, obeying operator priorities and grouping.

Combinatorial models such as reliability block diagrams or fault trees, assume stochastic independence between the components: the failure or repair of

a component is not affected by other components. If there is a need to model more complex interactions, where a component failure influences behavior of other components, other kinds of models must be used. An example of such cenario is a load-balancer Web service with four active instances. If one instance fails, remaining three will have higher load which will likely impact their reliability. This requirement cannot be modeled using combinatorial approaches described so far. One possibility that can be used to cover this class of problems are Markov models. Here we give a procedure to generate Markov availability models based on the Boolean equations:

- Let n be the cardinality of the set of all distinct terms in the minimal Boolean equation.
- The list $processedConfigurations$ is empty.
- The Boolean equation is parsed from left to right in order to generate set of states:
  - If two or more terms are connected with operator $\|$ and their configuration is not in $processedConfigurations$, new state variable is added to the model, and the variable can take the values of $\{0, 1, ..., k\}$, where k is the number of terms in the configuration and $k < n$. In other words, parallel elements generate states which take the value of the number of functioning units. The configuration is added to $processedConfigurations$.
  - If two terms are connected with operator & and the terms are not in $processedConfigurations$, a state variable is generated that can take values $\{0, 1\}$. Therefore, for each term from the sequential configuration, a state variable is generated which denotes if the term is up or down, and the term is added to $processedConfigurations$ list.
- The set of states is generated by:
  - Creating a state vector $V(v_1, v_2, ..., v_p), p < n$, v_i are the state variables and p is the number of generated state variables in the previous step.
  - Creating all possible states by allowing state variables to take all possible values from the sets defined in previous steps (parsing Boolean equation).
- Now state transitions are generated using given transition probabilities (MTTF, MTTR).
- The transition set is minimized using following rules:
  - If the coverage is imperfect (coverage exists), transition probabilities are multiplied with the coverage factor.
  - If repair priority is included in the model, only the higher priority transition is generated, that is, transitions with lower priority are eliminated.

– If limited repair resource is present, only so many transitions are generated as there are repair facilities available.

In the following sections we will demonstrate the proposed approach. We first show how to assess availability of an e-mail service and how far can we go with the assessment if the required data is only partially available (e.g., a service description exists, but infrastructure topology is partially known or unknown). Then, the approach is generalized to a business process from the publishing and media sector, demonstrating effects of technical infrastructure and personnel to overall availability. First two examples use combinatorial methods (RBD). Finally, Markov model generation will be demonstrated for the composite financial service with repair priorities.

5.4 E-Mail Service Availability Assessment

The proposed approach is demonstrated on the example of steady-state user-perceived availability assessment of the e-mail service. The example is based on SMTP protocol defined in RFC 2821 [117]. The e-mail service is chosen because of its ubiquity and presence in almost every modern enterprise. Availability of the e-mail service is evaluated for two users in order to show that availability is not only the function of component availability but also of infrastructure topology and user location within the topology.

5.4.1 Service Description and Mapping

The first step in availability estimation is to define the service of interest and its required availability. Let us assume that required availability of the e-mail service should be above 0.9985. BPMN service description is given in Figure 5.5. Service description is then mapped to existing infrastructure elements. CMDB provides the infrastructure graph (Figure 5.6) and component availability statistics (Table 5.1).

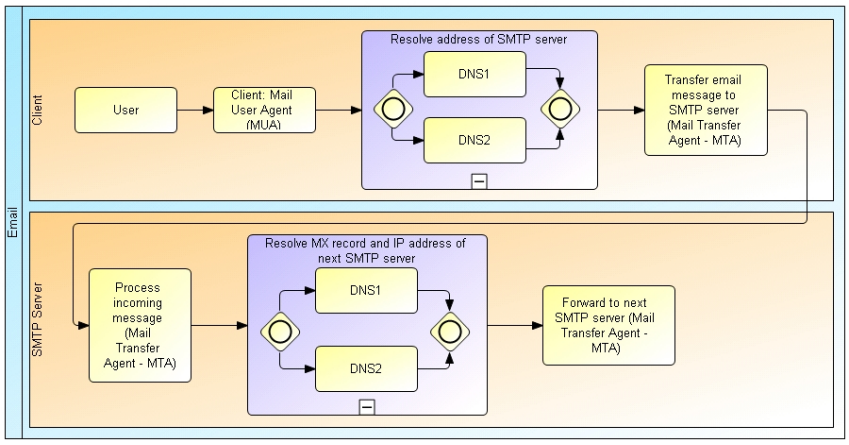

Figure 5.5: E-mail service description

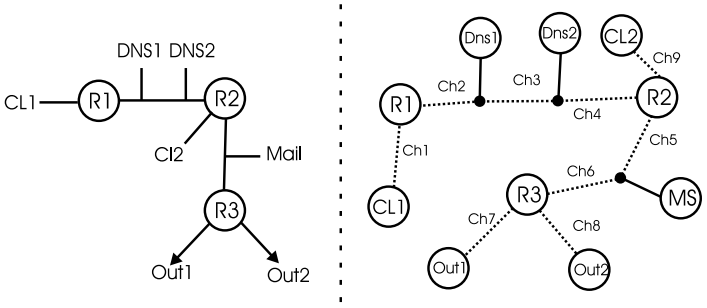

Figure 5.6: Infrastructure graph and transformation to connectivity graph

Abbreviations have the following meaning: Client i - CL_i, Mail Server - MS, Routers - R_i, Channel - CH_i, Out_1 and Out_2 are connections to Internet service providers of the enterprise. The channel is an abstraction that includes switches and/or network adapters/links that are placed between routers and hosts. They are introduced for simplicity reasons – the goal of this example is to demonstrate the approach without going into unnecessary details. In the next section an example is given with much more accurate and detailed network topology graph. Channels are depicted in Figure 5.6 by dotted lines. Based on the BPMN description from Figure 5.5 and connectivity graph from Figure 5.6, we map service execution steps to paths in the connectivity graph. In order to send an e-mail, client has to resolve address of the mail server. It is common that hosts in a network use two DNS servers, primary and secondary:

$CL_1 \rightarrow DNS:$
$(CL_1\&CH_1\&R_1\&CH_2\&DNS_1) \parallel (CL_1\&CH_1\&R_1\&CH_2\&CH_3\&DNS_2) =$
$CL_1\&CH_1\&R_1\&CH_2\&(DNS_1 \parallel (CH_3\&DNS_2)$
and
$CL_2 \rightarrow DNS:$
$(CL_2\&CH_9\&R_2\&CH_3\&CH_4\&DNS_1) \parallel (CL_2\&CH_9\&R_2\&CH_4\&DNS_2) =$
$CL_2\&CH_9\&R_2\&CH_4\&(DNS_2 \parallel (CH_3\&DNS_1)$

The clients now establish connections with SMTP server:

$CL_1 \rightarrow MS:$
$CL_1\&CH_1\&R_1\&CH_2\&CH_3\&CH_4\&R_2\&CH_5\&MS$
$CL_2 \rightarrow MS:$
$CL_2\&CH_9\&R_2\&CH_5\&MS$

In case that e-mail recipient is within the enterprise, following steps would not be performed and the e-mail message would be stored directly to disk system by the SMTP server, waiting there for local client to access it. In this example, we assume that a recipient is outside the enterprise and local SMTP server has to determine the forward SMTP server. This requires a DNS query:

$MS \rightarrow DNS:$

$(MS\&CH_5\&R_2\&CH_4\&DNS_2) \parallel (MS\&CH_5\&R_2\&CH_4\&CH_3\&DNS_1) = MS\&CH_5\&R_2\&CH_4\&(DNS_2 \parallel CH_3\&DNS_1)$

The last step is to dispatch e-mail to the outside server. Since we can neither measure nor influence availability of the Internet and outgoing (receiving) SMTP server, we evaluate availability up to the point where e-mail leaves the enterprise network:

$MS \rightarrow OUT:$
$(MS\&CH_6\&R_3\&CH_7\&OUT_1) \parallel (MS\&CH_6\&R_3\&CH_8\&OUT_2) = MS\&CH_6\&R_3\&(CH_7\&OUT_1 \parallel CH_8\&OUT_2)$

For the successful e-mail service execution, all these steps must be performed in series. The resulting expressions are simplified by applying idempotence, associativity and distributivity rules of operators & and $\parallel$:

$CL_1 : (CL \rightarrow DNS)\&(CL \rightarrow MS)\&(MS \rightarrow DNS)\&(MS \rightarrow OUT) = CL_1\&MS\&R_1\&R_2\&R_3\&CH_1\&CH_2\&CH_3\&CH_4\&CH_5\&CH_6\& (DNS_1 \parallel DNS_2)\&(CH_7\&OUT_1 \parallel CH_8\&OUT_2)$

$CL_2 : (CL_2 \rightarrow DNS)\&(CL_2 \rightarrow MS)\&(MS \rightarrow DNS)\&(MS \rightarrow OUT) = CL_2\&MS\&R_2\&R_3\&CH_9\&CH_4\&CH_5\&CH_6\&(CH_3\&DNS_1 \parallel DNS_2)\& (CH_7\&OUT_1 \parallel CH_8\&OUT_2)$

5.4.2 Availability Assessment

Boolean expressions that were generated can be directly transformed into fault trees (FT) or reliability block diagrams (RBD). For demonstration purpose, we have chosen to use RBD (Figure 5.7). The RBD is obtained by transforming & operator into serial configuration and operator $\parallel$ into parallel configuration. Evaluation parameters are in Table 5.1. The model was solved in Isograph Reliability Workbench and it assumes exponential distribution for failure and repair processes. The failure and repair rates are constant and they are calculated from MTTF and MTTR. Evaluation results are given in Table 5.2.

Table 5.1: Evaluation Parameters

	Router	Channel	DNS	Mail	Client	Out1	Out2
MTTF	9000	45000	4500	4000	4500	13500	5400
MTTR	1	3	2	2	2	4	6

Table 5.2: Evaluation Results

	Client1	Client2	Lower	Intermediate	Upper
MTTF	1060	1280	662	1240	6.1 e+22
MTTR	1.79	1.83	2.37	1.59	0.293
Unavailability	0.00166	0.00142	0.00322	0.00127	3.5 e-24

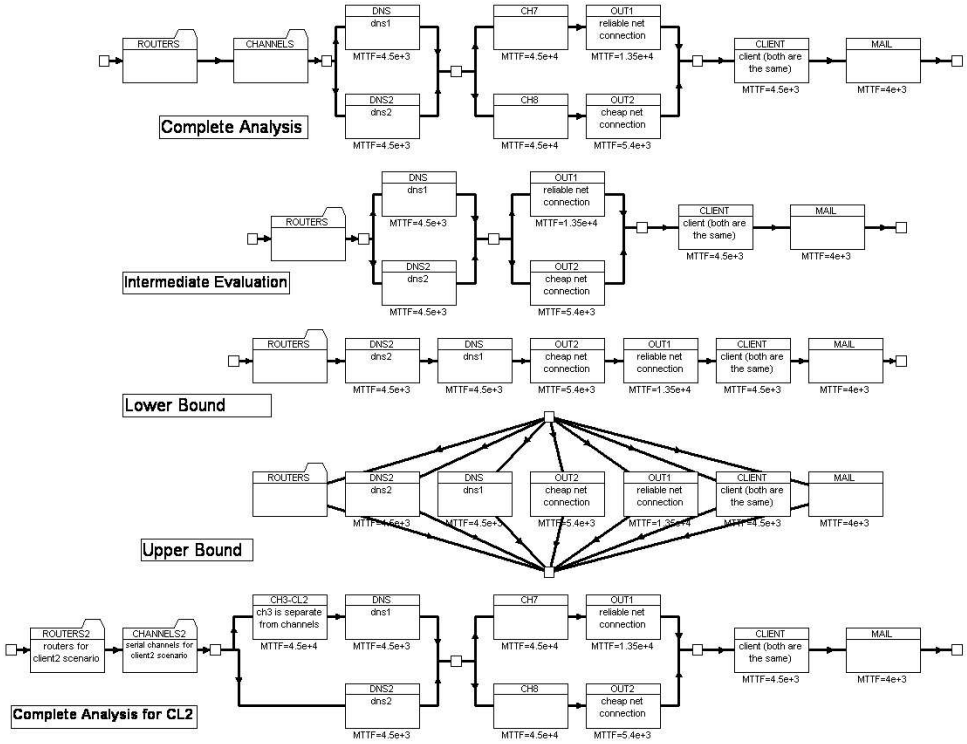

Figure 5.7: Reliability block diagram for the e-mail service

Required and user-perceived provided availability can be compared now. Provided availability of client 1 (calculated as 1 - unavailability) is 0.99834, and provided availability of client 2 is 0.99858. As our required availability is 0.9985, it is clear that e-mail service does not provide required availability to client 1.

Since measurement-based methods are already used to evaluate availability of individual infrastructure elements, like routers or servers, it could be tempting to claim that the same, measurement based approach should be used for service availability. However, as the user-perceived availability is network topology dependant and differs from one client host to another, it implies a monitoring application should be installed on every client host in the network for every monitored service. The overhead introduced through installation and maintenance of monitors on each client host, for every service the client is using, would be rather extensive and not very practical. Furthermore, for some IT services, such as e-mail where responsibility for service execution is delegated through the network, it is not straightforward to estimate availability by counting the success rates since measurements at individual points (server or client) ignore unavailability introduced by other infrastructure elements. If e-mail service success ratio is measured on the client-side only, the availability monitor cannot detect events where e-mail cannot leave the server because the Internet connection is not functional. Similarly, if the monitor is placed on a SMTP

server only, it is not able to detect events when client cannot connect to the server due to network failure. Therefore, precise service availability assessment through measurement requires careful monitoring of progress of individual e-mails through the whole IT infrastructure (outgoing e-mail is served once it leaves the enterprise, incoming e-mail once it reaches a client).

Our approach requires less effort for maintenance and provides additional advantage: in case of planned changes in the IT infrastructure, the impact of changes on availability can be estimated prior to implementation. For instance, if DNS1 server is moved to the same sub-network as the mail server, availability of the first client remains the same but availability of the second client increases to 0.99865. Pure measurement-based approach is not able to predict the impact of infrastructure changes on availability, before the actual change takes place.

5.4.3 Total Service Availability

The user-perceived availability of each client is not equivalent to total service availability. Given that we have already calculated steady state or interval service availability A_i for each client i, we investigate how to derive total service availability.

Service availability is the mean value of the user perceived availabilities (for all clients):

$$A_S = \frac{\sum_{i=1}^{n} A_i}{n} \tag{5.1}$$

where n is the number of clients (users) and A_i are either steady state or interval user-perceived availabilities.

Equation 5.1 assumes that all clients use the service equally. It can be made more precise by weighing it with usage factors:

$$A_S = \sum_{i=1}^{n} A_i \cdot u_i \tag{5.2}$$

Usage factor u_i, for client i, is the number of service invocations made by client i over the total number of service invocations. From its definition, it holds that the sum of usage factors for all users (clients) is equal to one.

For steady-state service availability, A_i is steady state user-perceived availability. Parameters u_i are calculated during the service's lifetime. In practical terms, this means that they are determined using statistical methods over a longer period of time.

If equation 5.2 is used to calculate interval service availability, A_i represents interval user-perceived availability. For interval availability, u_i is recorded for the observed interval, within which service availability is to be derived.

The impact of parameter selection to total availability is illustrated in table 5.3. As the usage factor of CL_1 is increased, service availability decreases since client 1 has lower steady state user-perceived availability. This demonstrates how usage factors balance total service availability in case where clients don't invoke the service or access underlying resources evenly.

Table 5.3: E-mail service availability for different values of parameter u_i. $A_{CL1} = 0.99834$, $A_{CL2} = 0.99858$

u_1	u_2	A_s
0.5	0.5	0.99846
0.6	0.4	0.998436
0.9	0.1	0,998364
0.1	0.9	0,998556

5.4.4 Working with Incomplete Data

The implicit assumption of the proposed method is that complete network topology, as well as availability of individual components, are known. Although many methods for determining service availability make this assumption, in practice this is frequently not the case. In such situations it is possible to estimate availability. Several cases of incomplete data can be distinguished.

Incomplete service description or network topology.

If service description (functionality) or the network topology are unknown, but ICT-components on which the service depends are known as well as their availabilities, the lower availability bound can be determined assuming that all components are placed in series:

$$LOWER : CL\&MS\&DNS_1\&DNS_2\&R_1\&R_2\&R_3\&OUT_1\&OUT_2$$

This model is unaware of communication channels and it does not include them. Similarly, the upper availability bound can be estimated assuming that all elements are placed in parallel:

$$UPPER : CL \parallel MS \parallel DNS_1 \parallel DNS_2 \parallel R_1 \parallel R_2 \parallel R_3 \parallel OUT_1 \parallel OUT_2$$

Finally, if the service description and component availabilities are known, but the exact network topology is unknown, availability can be estimated as:

$$INTERMEDIATE : CL\&MS\&(DNS_1 \parallel DNS_2)\&R_1\&R_2\&R_3\&(OUT_1 \parallel OUT_2)$$

The approximate service availabilities are given in table 5.2. The upper availability bound is of no practical use since it is very close to one. Lower bound is considerably lower than the actual availability, as expected. Intermediate model slightly overestimates availability but it is very close to precise values, considering that it does not utilize network topology information. Still, this particular intermediate model example should be taken with caution since the difference may be much larger for other, more complex infrastructure configurations.

The amount of knowledge that we have about the network and the service

Table 5.4: Fussel-Vesely Metric for different levels of system knowledge

	CL	MS	DNS1	DNS2	OUT1	OUT2	R1	R2	R3	CH
Complete	0.266	0.3	11.7e-4	11.7e-4	15.8e-4	18.3e-4	0.067	0.067	0.067	0.039
Lower	0.133	0.149	0.133	0.133	0.0818	0.271	0.033	0.033	0.033	/
Interm.	0.347	0.39	15.3e-4	15.3e-4	19.3e-4	19.3e-4	0.087	0.087	0.087	/

influences other metrics of importance, apart from availability estimation. The Fussell-Vesely [82] metric determines the probability that particular component has contributed to the system failure, given that the system has failed. The metric is important since it provides guidelines to network administrators where to incorporate the potential improvements in order to obtain the largest availability increase. Table 5.4 shows the differences observed in Fussell-Vesely metric for different levels of system knowledge:

- If complete information is known, according to FV metric, the administrator should improve availability of components in the following order: mail server, client, channels, routers. Other elements have minor impact on availability.

- If service description is known but network topology is unknown, according to FV metric, the administrator should work in the following order: mail server, client, routers. The fit between this and the precise evaluation is good and cannot mislead the administrator.

- If neither service nor topology of the network are known, based on the lower-bound assessment, the administrator should make improvements in the following order: internet connection 2, mail server, DNS servers, client, routers, internet connection 1. In this case, the metric is completely misleading and can distract the administrator from the real root cause of the problem: precise analysis has shown that DNS and outbound connections have minor impact on the e-mail service availability, yet the metric recommends that both of them should be checked with high priority.

Unknown availability of some components in the network

In case that it is not possible to determine availability of one or more components in a network that are used by the evaluated service, clearly, the exact service availability cannot be calculated. Assuming that availability is unknown for n components, availability can be observed as n-variable function and can be evaluated in n-dimensional space. It is necessary to assume the component availability distribution type or to take a distribution based on previous experience (e.g., if the availability distribution for one router type is known, in absence of better data it is to be expected that other routers from same producer will expose similar behavior), to vary distribution parameters and to observe availability. This approach is applicable to precise and approximate models, but it is highly dependent on human experience and actual behavior of unknown components.

Quantitative system data does not exist

It is sometimes required to make availability assessment even if we are unable to determine/measure IT component availabilities, services are not described and network topology is unknown. In such extreme conditions, our and other analytical or simulation based approaches are not applicable. One possibility is to use qualitative assessment, based on the best-practice guides like CobiT [105], ITIL [3], or BITCOM [2]. The best practices cover various aspects of IT management, therefore it is necessary to extract segments that are of importance for availability, clearly define questions, interview the personnel in the enterprise and finally interpret the answers. The interpretation can be quantitative or qualitative:

- Quantitative: [2] lists the expected downtime per year in data centers as the function of environmental factors. For example, if a data center has no redundant power supplies for equipment and air-conditioning, and no power generator, it can be expected that it may experience more than 72 hours of unplanned downtime per year. Another example is the CobiT process DS1 (Deliver and Support) that defines a metric that gives the percent of users satisfied with service delivery levels. As the CobiT specification does not define how to measure this percentage, the metric requires careful interpretation.
- Qualitative: Existence of formally defined RACI (Responsible, Accountable, Consulted, Informed) charts [105] clearly improves information flow in an enterprise and increases service availability. However, it is not possible to quantify availability improvement.

Best practices can be promptly implemented, providing coarse guidelines where to aim for availability improvement. Still, they are imprecise in comparison with analytical and simulation methodologies.

5.5 Publishing Business Process Availability Assessment

In this section, availability will be exemplary assessed for a business process from the publishing and media sector, which describes the scenario of accepting, processing and approving a new manuscript (the business process is shown in Figure 5.8).

5.5.1 Business process description and mapping

The process is initiated when the editor receives a new manuscript. She then initiates editorial tasks (art, marketing and finances) and delegates operational procedures to junior editors, which perform their tasks concurrently. Editor waits until all tasks are completed, evaluates the results, and makes a decision whether to approve the manuscript for printing or to return corrections. Once the editor is satisfied, manuscript goes to print and the business process ends.

Each activity of the business process is enacted by one or more services. Figures 5.9, 5.10, and 5.11 describe the following activities (composite services) of the business process respectively: initiating editorial tasks, junior editors' tasks and evaluation of the junior editorial-tasks (acceptance/rejection) by the editor.

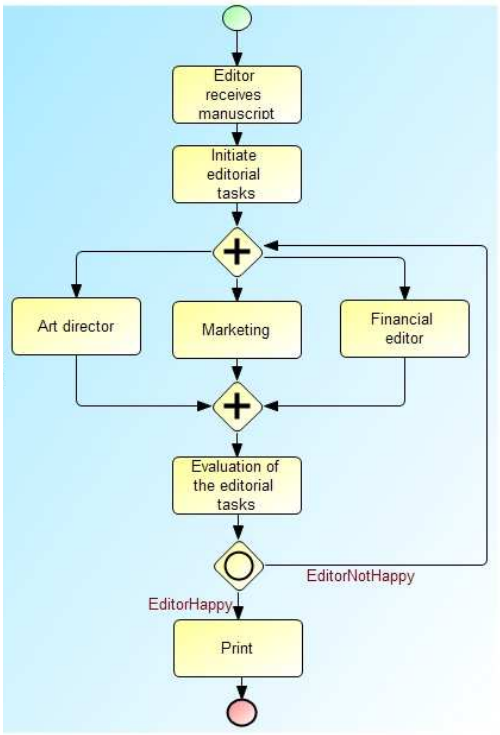

Figure 5.8: Business process describing acceptance of a new manuscript

The following atomic services are available to the business process: Generate Ticket, Access Network Disk, Receive Ticket, Prepress, Create Marketing Plan, and Create Financial Plan. In order to initiate editorial tasks, the editor generates tickets (addressed to junior editors) and uses network disk access service to make the manuscript available to junior editors. Junior editors use the read ticket service to receive tasks, access network disk service to obtain the manuscript copy, and then process the manuscript (prepress, marketing and financial plan services). Parallel to these activities, they repeatedly access the network disk. Finally, all junior editors generate tickets once they have finished their tasks. In order to evaluate the results, editor has to receive tickets from all junior editors, access network disk to inspect the results, and generate the ticket, either to approve the print or to return relevant corrections. BPMN activities that are denoted with the UML use case actor symbol imply that manual activity is performed at that point in the workflow.

The business process is executed on the infrastructure shown in Figure 5.12. It is assumed that infrastructure data is provided by a CMDB system (step 3 of the algorithm). Routers $R1$ to $R3$ form the core of the network. Subnetworks are connected to the core via switches $S1$ to $S7$. Subnetworks are divided by

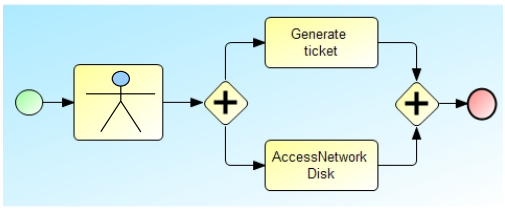

Figure 5.9: Initiating editorial tasks service

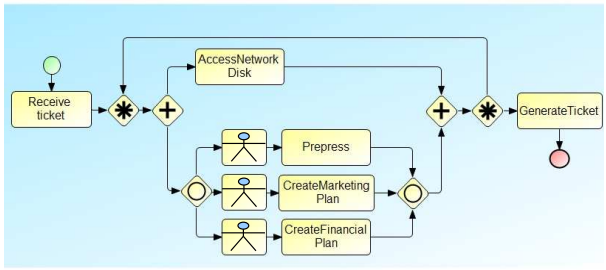

Figure 5.10: Junior-editors task service

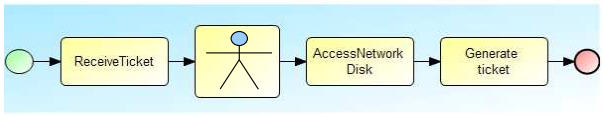

Figure 5.11: Editorial tasks evaluation service

intended usage: there is a subnetwork for editor's (CL_e) and directors' (financial CL_f, marketing CL_m and art CL_a) client computers, a subnetwork for other employees, for DNS and web servers, etc. Other infrastructure components important for availability assessment are ticket server TS, file server FS, financial server SAP, and server running graphical applications (print prepare) PP. The core routers are placed at considerable distance and cables ($C1$ to $C3$) interconnecting them are included in the availability evaluation as they are more likely to be damaged and their repair time is non-negligible. Cabling within local subnetworks is considered highly available (unlikely to be damaged, rather easy to diagnose and repair) and ignored in this analysis. In case that local cabling is also considered to be of high relevance for availability, it can easily be added to the evaluation process.

The first task in step 4 of the algorithm is to describe individual communication paths that are used by atomic services: e.g., access of a client computer to DNS server. In order to successfully communicate, initiator (source), executor (destination) and the infrastructure between them need to be active and functional. Atomic services can be also described directly, but derivation of equations for communication paths simplifies that task and encourages reuse of equations - typically each communication path is used by multiple services. The key communication paths are:

- Editor's client computer to DNS:

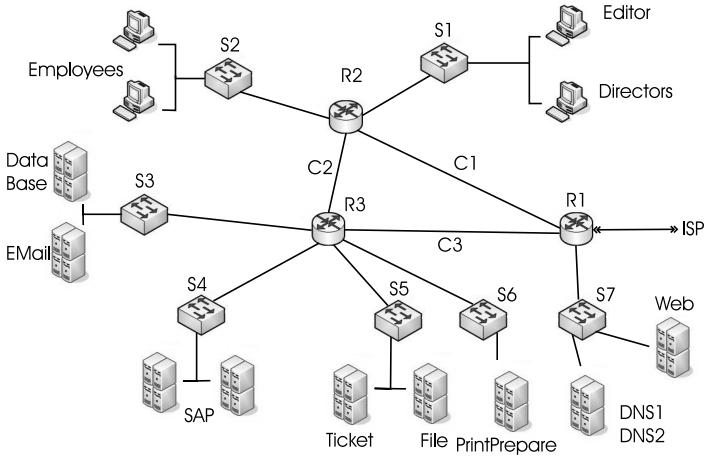

Figure 5.12: Infrastructure/communication graph for the publishing business process

$$CL_e \rightarrow DNS : CL_e\&S1\&R2\&(C2\&R3\&C3||C1)\ \&R1\&S7\&(DNS1||DNS2)$$

- Editor's client computer to ticket server:
  $CL_e \rightarrow TS : CL_e\&S1\&R2\&(C2||C1\&R1\&C3)\ \&R3\&S5\&TS$
- Financial director's client computer to financial server:
  $CL_f \rightarrow SAP : CL_f\&S1\&R2\&(C2||C1\&R1\&C3)\ \&R3\&S4\&SAP$
- Art director's client computer to print preparation server:
  $CL_a \rightarrow PP : CL_a\&S1\&R2\&(C2||C1\&R1\&C3)\ \&R3\&S6\&PP$
- Ticket server to DNS:
  $TS \rightarrow DNS : TS\&S5\&R3\&(C2\&R2\&C1||C3)\ \&R1\&S7\&(DNS1||DNS2)$
- Ticket server to editor's client:
  $TS \rightarrow CL_e : TS\&S5\&R3\&(C2||C1\&C3\&R1)\ \&R2\&S1\&CL_e$
- Print preparation server to file server:
  $PP \rightarrow FS : PP\&S6\&R3\&S5\&FS$
- Financial server to file server:
  $SAP \rightarrow FS : SAP\&S4\&R3\&S5\&FS$

For brevity, we do not write down all communication paths. Omitted paths can be easily generated: for instance, access of art director's client computer to DNS servers is essentially the same as for the editor's client as they belong to the same subnetwork. Of course, for evaluation of the service/business process availability, all activities have been included. Equations describing atomic and composite services are derived iteratively from the description of communication paths, repeating step 4 of the mapping algorithm. If a service has more than one user (e.g., generate ticket service), the service user is marked in superscript (e.g., editor's generate ticket service is marked as GT_s^e while art director's generate ticket service is marked as GT_s^a). The atomic services are:

- Generate Ticket service:
  $GT_s^e : (CL_e \rightarrow DNS)\&(CL_e \rightarrow TS)$
- Access Network Disks service:
  $AND_s : (CL_e \rightarrow DNS)\&(CL_e \rightarrow FS)$
- Ticket Reception service is initiated by the ticket server: $TR_s : (TS \rightarrow DNS)\&(TS \rightarrow CL_e)$
- Print Preparation service, after initiation by the arts director is executed on PreparePrint servers:
  $PP_s : (PP \rightarrow DNS)\&(PP \rightarrow FS)$.
- Financial Calculation service
  $FC_s : (CL_f \rightarrow DNS)\&(CL_f \rightarrow SAP)\&(SAP \rightarrow DNS)\&(SAP \rightarrow FS)$.
- Marketing Plan service
  $MP_s : (CL_m \rightarrow DNS)\&(CL_m \rightarrow SAP)$.

In addition to ICT-components, persons (editor ED, art director AD, financial director FD and marketing director MD, all marked with UML actor symbol) are involved in the composite services execution:

- Initiate Editorial Tasks composite service (Figure 5.9)

$$IET_s : ED\&AND_s^e\&GT_s^e \tag{5.3}$$

- Junior-editor Tasks composite service (Figure 5.10)

$$JET_s : (TR_s^a\&AD\&AND_s^a\&PP_s\&GT_s^a)\&$$
$$\&TR_s^f\&FD\&AND_s^f\&FC_s\&GT_s^f)\&(TR_s^m\&MD\&MP_s) \tag{5.4}$$

- Evaluation of the Editorial Tasks composite service (Figure 5.11)

$$EET_s : TR_s^e\&ED\&AND_s^e\&GT_s^e \tag{5.5}$$

Finally, composite services are used to describe availability of the business process (step 5 of the mapping algorithm):

$$Print : IET_s\&JET_s\&EET_s. \tag{5.6}$$

Expressions 5.3, 5.4 and 5.5 are substituted in 5.6 and then further expanded with atomic service and communication path descriptions. At the end, mapping of the business process on the ICT-component layer is obtained as follows:

$$ED\&CL_e\&AD\&MD\&FD\&CL_a\&CL_f\&CL_m\&FS$$
$$\&TS\&SAP\&PP\&(DNS1||DNS2)\&S6\&S4\&S1\&S7$$
$$\&S5\&R1\&R2\&R3\&((C1\&(C2||C3))||(C2\&C3))\&INF \tag{5.7}$$

5.5.2 Business Process Availability Assessment

In the step 6 of the algorithm formal availability model is created from equation 5.7. We transform it to reliability block diagram (RBD) using the rule that & symbol is transformed to serial while $\|$ is transformed to parallel connection of components. Figure 5.13 shows the RBD corresponding to equation 5.7.

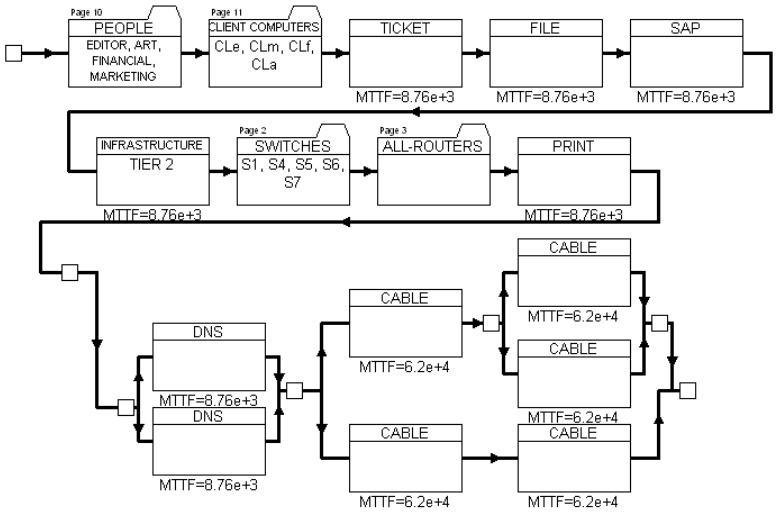

Figure 5.13: RBD corresponding to the business process from figure 5.8

RBD from Figure 5.13 is solved using an external tool in the step 7. We have used Isograph Reliability Workbench. Table 5.5 shows availability parameters of individual infrastructure components. The parameters are taken from industrial studies performed by Yankee group [4] and Schweitzer Engineering Laboratories [179]. We assume that DNS and file servers operate on RedHat Linux, SAP on AIX, personal client computers are running Microsoft Windows (due to unavailability of data for Windows desktop systems, we are using data for Microsoft Server 2003), print prepare runs on Solaris. For cabling we had to use our own estimation based on the various sources, interviews with technical staff and personal experience. For routers and switches we use data from [179] assuming that enterprise uses standard routers. In solver, failure and repair rates are modeled as exponentially distributed.

Table 5.5: Evaluation Parameters (in hours)

	Routers	Switches	Cables	DNS, FS, TS	SAP	Clients	PP
MTTF	83220	100740	62000	8760	8760	8760	8760 0
MTTR	48	48	3	1.73	0.6	8.9	1.44

Human errors are not described by the exponential failure distribution, like it is common for the IT components. Instead, they have fixed failure rate. The data for errors introduced by human operators is taken from [116] for trained personnel. The impact of human errors to availability is set to 0.1

Observing hardware, software and network availability ignores the impact

	IET_s/EET_s	JET_s	BP
IT only	0.0027	0.0049	0.0059
IT + Infrastructure	0.0052	0.0074	0.0084
IT + People	0.0037	0.0079	0.0099
IT + People + Inf.	0.0062	0.0104	0.0124

Table 5.6: Evaluation results – unavailability

of supporting infrastructure that, according to numerous industry experiences, has significant impact on system availability. Uptime Institute has defined four classes (tiers) of infrastructure that define additional downtime that supporting infrastructure adds to IT systems [205]. In our example we assume that publishing company running the business process has implemented tier 2 infrastructure: the redundancy of power supply, UPS and air conditioning is limited, while maintenance of critical parts of the infrastructure requires processing shutdown. According to [205], tier 2 infrastructure INF increases system downtime by 22h annually.

Table 5.6 shows evaluation results of steady-state availability analysis for composite services and the business process. Four cases are investigated, differentiating whether the impact of people and support infrastructure is included in the evaluation.

If it is assumed that people and support infrastructure are ideal ('IT only' entry in the table), availability of the business process is 0.9941. More precise models that include human (IT + people) or supporting infrastructure (IT+infrastructure) failures show decrease in availability to 0.9901 and 0.9916 respectively. Failures that are introduced jointly by humans and support infrastructure double unavailability, reducing availability of the business process to 0.9876.

5.6 Generation of State Space Models

Some scenarios cannot be modeled using combinatorial method such as RBD or FT. In such cases it is necessary to generate an adequate state space model, based on the proces/service description and the infrastructure graph. Let us observe a service for writing invoices shown in Figure 5.14.

An invoice is created and sent to financial assistance services, which calculate appropriate taxes, based on the invoice type and concent. There are two such services working in redundant configuration. They read invoices, calculate tax and then the first answer is accepted (completed invoice) and written in the data store. Furthermore, it is allowed for one financial assistance service to fail, and the system is still considered operational. The overall service fails if write invoice operation fails or both tax calculation services fail. Repair priority is also defined, such that the repair of invoice writing operation always has higher priority over tax calculation repair. This information may originate from GSTool as shown in Figure 5.15 or it may be manually provided by the

user. Due to prioritized repair process, this process cannot be modeled using combinatorial methods. Instead, we will demonstrate how to generate Markov availability model.

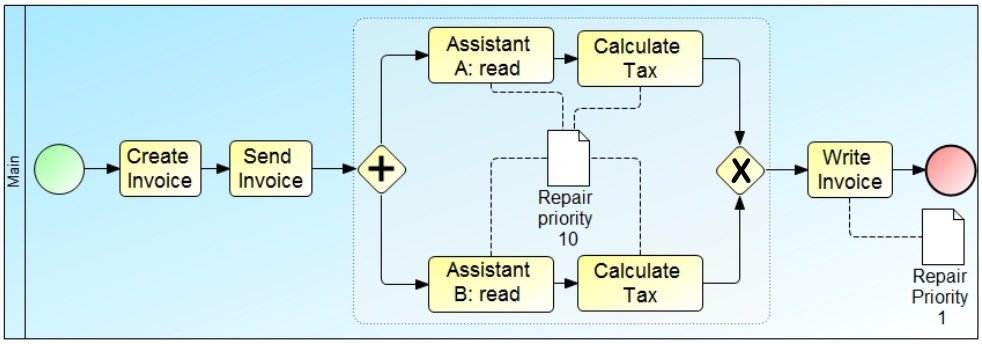

Figure 5.14: Service for writing invoices

Figure 5.15: Repair priority definition in GSTool

The infrastructure graph for the execution of this service is given in Figure 5.16. In this scenario, Financial Service C, as well as redundant data storage are not used (we will be needing them for the next section where we discuss hierarchical models).

Following the mapping procedure, we identify critical service communication paths:

- CreateInvoice: $User\&CIC$
- SendInvoice: $CIC\&C1\&S1\&R1\&S2\&(C2 \parallel C3)\&(FSA \parallel FSB)$
- CalculateTax: $FSA \parallel FSB$
- WriteInvoice: $(FSA\&C2) \parallel (FSB\&C3)\&S2\&R1\&R2\&S3\&C4\&DS$

The overal expression for the invoice service is CreateInvoice & SendInvoice & CalculateTax & WriteInvoice. Contrary to RDB/FT generation, the process of Markov model generation is much more complex. In order to simplify the example, let us assume without loss of generality, that all network systems are perfect and that the client computer (CIC) is also ideal (fault-free). Thus, availability expression for the service becomes: $(FSA \parallel FSB)\&DS$. In other words, the system will fail if and only if both financial services fail, or a datastore fails. In this case vector S will have two variables (v_1, v_2) where v_1 may take the values from $\{2, 1, 0\}$, describing the number of operating financial services, while v_2 takes the values from $\{1, 0\}$ describing if the datastore is functional or not. Markov model is now generated by creating all transitions between states

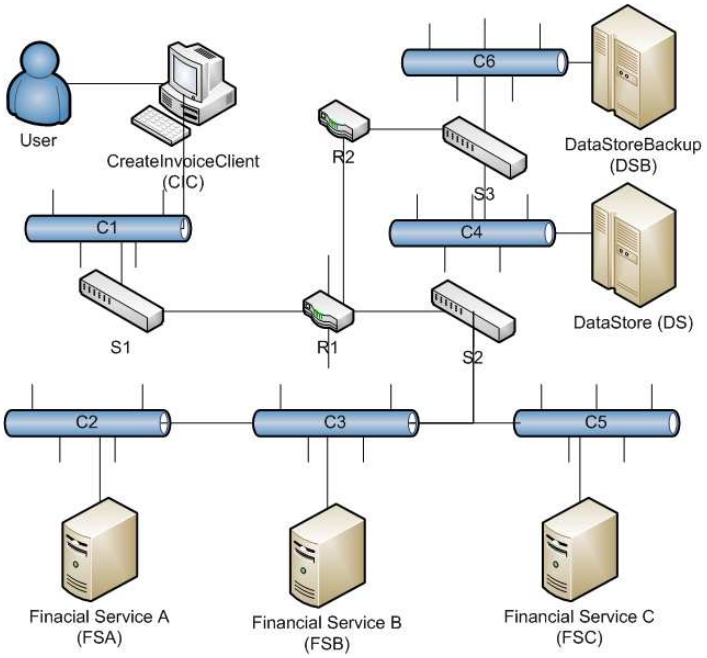

Figure 5.16: Infrastructure graph for the invoice service

described by all possible combinations of (v_1, v_2), taking into account repair priorities, that is, it is not allowed to repair financial service while datastore is down. Assuming that the failure and repair rates for both financial services are equal to λ_{FS} and μ_{FS}, and that the failure and repair rate of the datastore are λ_{DS} and μ_{DS}, the resulting Markov model is given in Figure 5.17. Note that, if we didn't assume that client and network are ideal, the corresponding Markov model would have 384 states, which is clearly not possible (or at least very difficult to) model manually and correctly at the same time. This is one of the major advantages of this approach – such complex models are generated automatically, based on the process description, regardless of the number of infrastructure elements. The assumption we made was for presentation purposes only, as it is not possible to visualize 384-state model in a meaningful way.

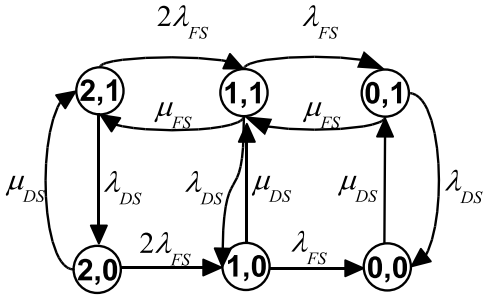

Figure 5.17: Markov model for the invoice service

The generator matrix of this system is:

$$\mathbf{Q} = \begin{bmatrix} -\lambda_{DS} - 2\lambda_{FS} & \lambda_{DS} & 2\lambda_{FS} & 0 & 0 & 0 \\ \mu_{DS} & -\mu_{DS} - 2\lambda_{FS}) & 0 & 2\lambda_{FS} & 0 & 0 \\ \mu_{FS} & 0 & -\mu_{FS} - \lambda_{DS} - \lambda_{FS} & \lambda_{DS} & \lambda_{FS} & 0 \\ 0 & 0 & \mu_{DS} & -\mu_{DS} - \lambda_{FS} & 0 & \lambda_{FS} \\ 0 & 0 & \mu_{FS} & 0 & -\mu_{FS} - \lambda_{DS} & \lambda_{DS} \\ 0 & 0 & 0 & 0 & \mu_{DS} & -\mu_{DS} \end{bmatrix}$$

Steady-state and instantaneous service availability is calculated as:

$$A_S = \pi_{21} + \pi_{11}$$

$$A(t) = \pi_{21}(t) + \pi_{11}(t)$$

The steady-state availability, mean time to failure, downtime and mean time to system restoration are given in table 5.7 and the transient analysis in Figure 5.18, with $MTTF_{FS} = \frac{1}{\lambda_{FS}}$=3000, $MTTR_{FS} = \frac{1}{\mu_{FS}}$=5, $MTTF_{DS} = \frac{1}{\lambda_{DS}}$=5000 and $MTTR_{DS} = \frac{1}{\mu_{DS}}$=2 (calculated with Sharpe).

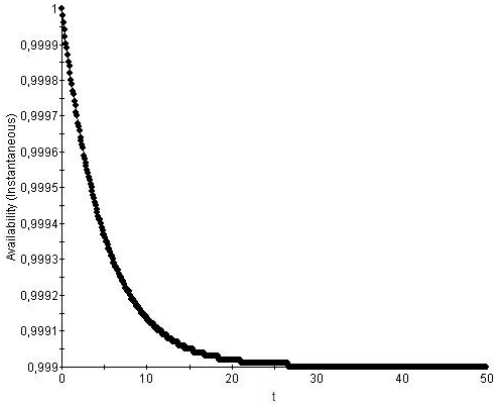

Figure 5.18: Instantaneous availability for the invoice service

A_S	$MTTF$	$MTTR$	$Downtime$
0.9990001	4988.91	4.99337	52.55431

Table 5.7: Analysis results for the invoice service

Two distinct differences between combinatorial and state-space models can be summarized in this example: the possibility to model a system where services have repair priorities and the assessment of interval (transient) availability, both of which are not possible with combinatorial models.

5.7 Generation of Hierarchical Models

As noted in the previous section, if the business process is extended, or the entire infrastructure (e.g., all network elements) is taken into account when performing availability analysis, the number of states in Markov models grows. Furthermore, state-space models are not easy to design (generate) and analyse. On the other side, there are problem aspects that can be solved only with state

space models. One possibility is to use hierarchical modeling, where certain problem aspects are described with combinatorial models, and other with state-space models.

Assume that the business process from the previous example is changed in such a way that another financial service is present (Financial Service C), and redundant data store is used (Data Store Backup). The process is further modfied, so that all three financial services are executed in parellel, and voting is performed on the outcome. In order for the voting to be successful, at least two services must succesfully complete. One datastore may also fail. Finally, this time we take into account one part of the network infrastructure, router $R1$, which is not perfect and may also fail. The expression characterizing this process is:

$$InvoiceModified : (FSA \parallel FSB \parallel FSC)_{2/3} \& R1 \& (DS \parallel DSB)$$

Up to this point, we could model this business process with combinatorial models, such as fault tree given in Figure 5.19.

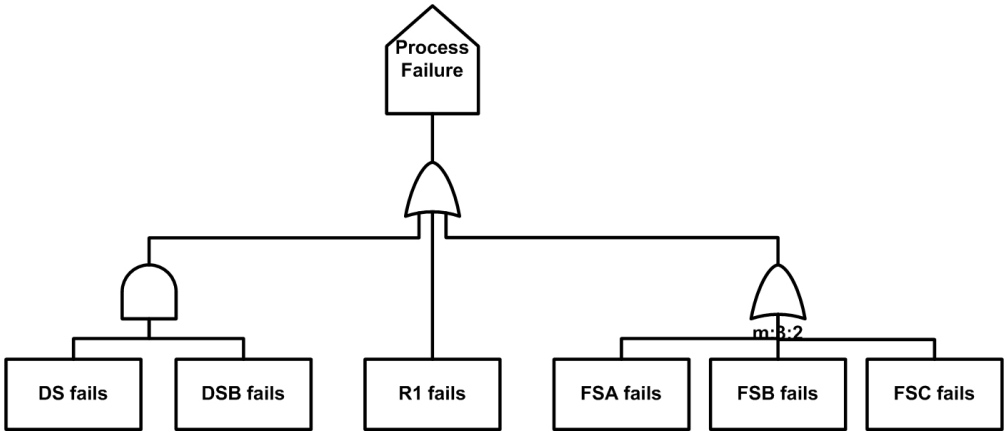

Figure 5.19: Fault tree for the modified invoice business process

This model, however, assumes independent repair facilities for each component. If we restrict this and allow for one repair facility per system type, that is, single repair facility for data store servers, single repair facility for financial servers and single repair facility for routers, this scenario cannot be modelled with any kind of combinatorial model. A solution is either to remodel the scenario using some state-space model, or to use hierarchical modeling, and to represent certain parts of the combinatorial model with state space models. This is the approach we adopt here and show how to perform nesting of Markov models inside a fault tree model.

For each subsystem whose behavior cannot be expressed using a combinatorial model, we derive a Markov model, as shown in Figure 5.20. Markov models simply express that there is one repair facility for data store, one for financial services, and one for routers. Based on the Markov models, availability, MTTF and MTTR of those subsystems can be calculated.

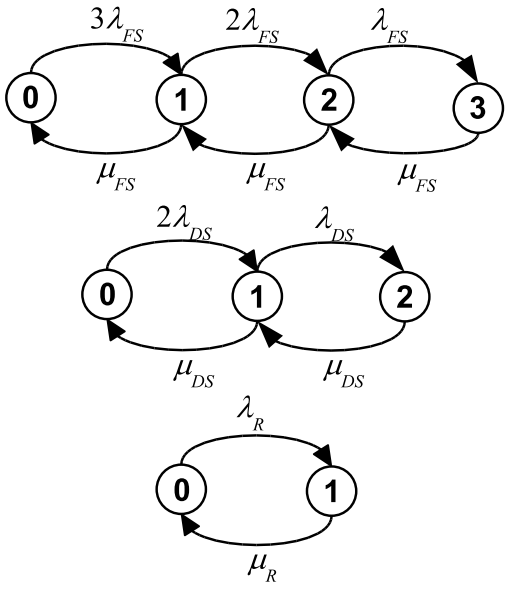

Figure 5.20: Markov models for the modified invoice process

Assuming that the mean time to failure of data store service is one month, of financial service two months and of router three months, and that all mean time to repair values are equal to 2.5 hours, after calculating subsystem availability using Markov models from Figure 5.20 and substituting it into the fault tree from Figure 5.19, the overall system availability is evaluated to 0.9961. If we used only combinatorial models, ignoring the limited repair capacities, the availability is 0.9995, which is a significant difference, as it represents more than one additional day of downtime per year. This rather simple example justifies the use of state-space models and the need for their application is even greater if complex services, that are deployed in enterprise-sized infrastructures, are to be studied.

5.8 Tool Prototype

In order to support business process and service mapping, a tool is currently being developed which enables the mapping of ICT-layer components to services and business processes, as well as calculation of service and process availability with automatic model generation. For that purpose, a model-based approach based on Meta Object Facility (MOF)[156] is used. The concepts of failure modes, availability and necessary transformations are described at the meta-model (MOF M2) level, while the instances are described at the model (MOF M1) level. MOF levels are shown in Figure 5.21.

The tool architecture is shown in Figure 5.22. The tool accepts service or business process description in high-level process language (currently BPMN and UML activity diagrams) and infrastructure data collected from a CMDB (currently OpenNMS) as input. This requires graphical BPMN/UML editor

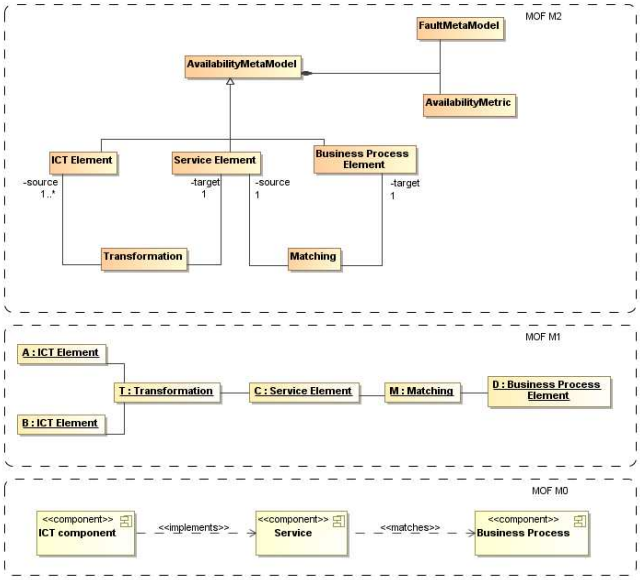

Figure 5.21: Model-based architecture for the mapping and availability assessment

and OpenNMS installation. The main tool is realized as Eclipse Rich Client Platform (RCP) application [74] using Eclipse Modeling Framework (EMF)[71] and Graphical Modeling Framework (GMF)[72]. It generates infrastructure graph, and enables (graphical) mapping of business process and service description elements to ICT infrastructure elements, which results in a connectivity graph. Based on the transformation rules specified in Atlas transformation language (ATL)[129], transformation is performed on the connectivity graph using Eclipse M2M project [73], transforming it into formal model description required for the external solver. Currently, transformations to reliability block diagrams, fault trees and Markov chains are supported. The resulting model is then used as input to an existing solver (currently Isograph Reliability Workbench and Sharpe), which computes provided business process and service availability. Required and provided service availability are then matched, and if discrepancies are found, adequate action can be taken.

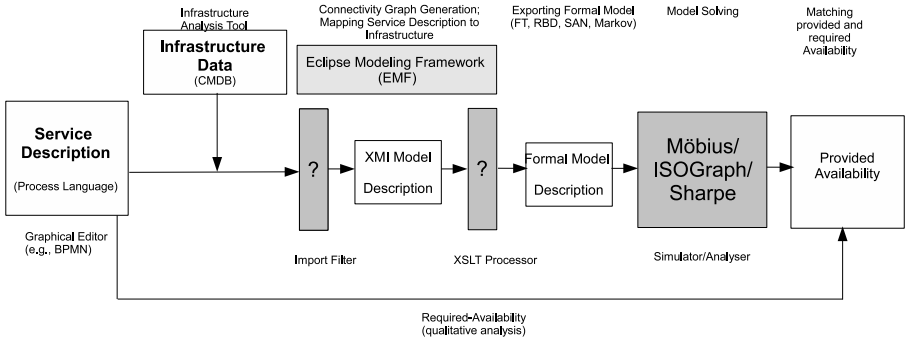

Figure 5.22: Mapping and availability assessment tool architecture

It is also planned to develop a repository supporting the tool which will store service and process templates, as well as information required for model parameterization (e.g., MTTR and MTTF or frequently used system components). The basic idea is to provide a comprehensive collection of availability data for the known components at the ICT-layer, similarly to the way that standard availability assessment modeling tools manage part catalogues such as MIL-HDBK-217 or Telcordia SR-332. For this purpose, however, a deeper understanding of the nature of the fault characteristics of component types other then hardware, primarily software, supporting infrastructure and particularly people (human faults), is required.

Chapter 6

Summary and Future Work

Service and process availability assessment and management are rapidly coming into focus of the IT operations research, due to difficult process requirements and interdisciplinary approach required for the successful solution. In this work we provided theoretical background, investigated tool applicability and reported our experience in the field of service availability assessment.

As both terms *service* and *availability* are often used rather colloquially and with different meanings in various areas and contexts, we introduced a framework in which we presented the problem of *service availability assessment*. In Chapter 2 we formally defined reliability and availability, and also discussed different perceptions of services. After introducing the reference architecture comprising the business process, service and ICT-component layers, we provided extensive definitions of different types of service and business process availability that we subsequently treat.

We then investigated existing analytical and qualitative availability models in Chapter 3 with the intention to examine their applicability to service availability assessment. Analytical models we covered include reliability block diagrams, fault trees, reliability graphs, Markov models, stochastic Petri nets, and stochastic activity networks. Furthermore, as we argued that services are often characterized by reduced operation capabilities or performance levels in the presence of faults, we presented Markov reward model which may be used to model service performability. Often, analytical models are difficult to apply due to scalability and parameterization problems. For that purpose, we investigated qualitative models (so-called reference or maturity models) that may be used for qualitative service availability assessment. We identified CMMI, ITIL, CITIL, CobiT, MOF, MITO as well as may standards (such as ISO/IEC 27002 or ISO/IEEE 12207) as potential sources of qualitative data on service availability.

Chapter 4 and Appendix B presented a detailed survey of more than 60 existing tools for availability assessment. Using one evaluation schema we compared features of tools from the following categories: analytical and simulation tools, benchmarking tools, risk management tools, process management tools and hybrid tools. We also discussed issues of tool interoperability and usability. Based on this part of the study, the following conclusion was drawn: existing

methods and tools are powerful, but difficult to apply directly for modeling service availability. Historical development of availability models and tools has associated them with mission-critical systems that rarely change during their lifetime, such as real-time systems, avionics, or telecommunication circuits. The availability assessment procedures they offer are unable to adapt to fast-paced changes in modern SOA systems. Each time the IT infrastructure or business process changes, it is required to manually intervene and update, verify and evaluate availability model. This is the consequence of outdated philosophy behind standard availability models – they were primarily introduced to facilitate static fault-tolerant system design, which comprises the following steps: model a system, design a system, evaluate a system and then forget the model. Standard models are also very complex and manually built. Therefore, only isolated and relatively small portions of complex SOA infrastructure can be feasibly and realistically modeled using state-of-the-art general purpose availability assessment models and tools. Furthermore, specialized and often highly mathematical knowledge and expertise is required to operate the tools, which is another reason for their low penetration in the service industry.

For these reasons, we propose automatic generation of service availability models as a potential solution to above mentioned problems. In Chapter 5 such a procedure is detailed. It is based on the premise that availability of services and business processes is influenced by the behavior of the underlying ICT-components which are used for their realization. Thus, a mapping procedure was defined which enables to determine how availability properties of complex business processes and services relate to availability properties of the underlying infrastructure (e.g., hardware, software, supporting infrastructure, network and personnel). After this mapping has been established, service availability model (such as reliability block diagram or Markov chain) can be automatically generated. We also presented several examples of service and process availability assessment performed using this method, and identified the benefits achieved such as independence from the particular availability model, reducing the level of expertise, the ability to automatically react to changes in the infrastructure and reevaluate the availability model as well as to simulate planned changes and perform return on investment calculation.

We see the proposed method as a first step to establish comprehensive model-based service management infrastructure, with the goal to complement existing measurement and empirical-based process management frameworks (e.g., IBM Tivoli or HP Mercury) and maturity models (e.g., ITIL or CobiT) with precise fault models and analytical capabilities. The elements of such service management roadmap are:

- *Automatic generation of new reliability/availability models.* To determine other properties of interest and to model more complex systems, additional models may be required, such as Petri nets or stochastic activity networks. Such additional models (and the corresponding solvers) should be dynamically integrated in the proposed infrastructure.

- *Model parameterization.* It is still not clear how to parameterize all elements of the ICT-infrastructure. Frequently it is not possible either to

measure or to obtain availability parameters of a system. Qualitative methods have to be investigated further which may help in estimating the missing parameters. It is not only the parameterization problem, but also more fundamental problem of choosing the adequate distribution to describe a physical occurrence such as operating system, custom application, network cable or human operator failure. In this work we used only exponential distributions and introduced fixed rate failures for human operators and infrastructure. We are currently experimenting with other distributions (such as Weibull, Erlang, normal, Gamma, Bayesian etc.) and applying them to different ICT-layer elements.

- *Monitoring tools.* Essential part of the solution for service management are elaborate runtime monitoring capability, such as those built into existing process management tools. Interoperability of such tools is, however, very limited. As the fundamental property of any SOA system is high dynamic and absence of central controlling entities, this drawback has to be overcome as the monitoring data from heterogeneous systems will have to be pulled together and aggregated in order to perform high-quality availability prediction.

- *Performability models.* Services do not necessarily fail right after an infrastructure failure has occurred, but may continue working in the degraded performance mode (e.g., reduced bandwidth, lower transaction throughput). This phenomenon and its consequences are captured by performability models, such as Markov reward models, which can be generated as additional output of the proposed framework.

- *Availability of Business to business (B2B) interactions.* The case studies we demonstrated in this work involved in-house business processes, described using orchestration. The issue of tool interoperability becomes even more important when choreographed business processes have to be evaluated, as frequently not enough data and/or control is available to perform qualified availability assessment. Upper and lower bound estimation methods, such as demonstrated for the E-mail service with incomplete topology knowledge, have to be developed to enable treatment of such scenarios. Furthermore, parameterization and modeling of best-effort environments, such as the Internet, also play important role in understanding availability properties of cross-enterprise processes.

- *Mapping of availability metrics to concrete business tasks and objectives.* It is important to understand properties such as SLA parameterization or calculation of costs that are imposed on an enterprise by service and business process unavailability. Coarse grain approaches such as [160], where costs of downtime are calculated under assumption that either all processes in an enterprise operate within their specifications or neither of them, can be considerably refined. For example, return on investment (ROI) analysis may be performed by simulating additional investments in the infrastructure and observing how changes in user-perceived service

availability impact the profit. Additionally, weak spot identification as well as business root cause analysis should be enabled, where technical measures (availability) are mapped into business measures, which can be further automatically included into decision making process.

- *Service Level Agreement (SLA) estimation.* One of the crucial issues for the providers of software as a service or cloud computing solutions is the ability to be able to determine the SLA level offered to the clients. The presented results enable analytical estimation of service availability, but may include other properties, such as security or timeliness. Such a framework which would, on the provider side, enable strict SLA parameterization, and on the client side, enable SLA measurement, monitoring and automatic adaptation, would prove beneficial for the acceptance of the incoming novel paradigms.

- Finally, *education in modeling, fault-tolerance and service-oriented architecture* areas has to be popularized in order to understand the acuteness of the problem facing still small community trying to address dependability aspects of modern service-oriented systems. Efforts such as conferences (e.g., International Service Availability Symposium) and organizations/consortiums (e.g., Service Availability Forum) are first steps in this direction. Furthermore, the proposed approach addresses availability assessment at different layers, providing availability awareness at different levels of enterprise management and giving a clear message which ICT components or services are of critical value for the overall business prosperity.

Bibliography

[1] Web Service Reliable Messaging Protocol. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-rm/ws-reliablemessaging200502.pdf, 2005.

[2] Betriebssichere rechnenzentren. BITKOM Consortium, 2006.

[3] IT Infrastructure Library. http://www.itil-officialsite.com, 2007.

[4] Global Server Operating System Reliability Survey 2007-2008. Yankee Group, 2008.

[5] Merriam Webster Online. *http://www.merriam-webster.com/dictionary/service*, 2008.

[6] GSTool Homepage. http://www.bsi.de/gstool/index.htm, 2009.

[7] RFC 3411. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. *http://www.ietf.org/rfc/rfc3411.txt*, 2002.

[8] E.A.P. Alchieri, A.N. Bessani, and J. da Silva Fraga. A dependable infrastructure for cooperative web services coordination. In *Proc. IEEE International Conference on Web Services*, pages 21–28, 2008.

[9] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[10] T.F. Arnold. The Concept of Coverage and Its Effect on the Reliability Model of a Repairable System. *IEEE Transactions on Computers*, C-22, 1973.

[11] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput*, 2004.

[12] P. S. Babcock, F. bong, and E. Gai. On the Next Generation of Reliability Analysis Tools. Technical report, 1987.

[13] K. Bachman. Surf-2 user guide. *http://www.laas.fr/surf/binary/surf2-doc.ps.Z*, 1996.

[14] M. Balakfushnan and C.S. Raghavendra. On Reliability Modeling of Closed Fault-Tolerant Computer Systems. *IEEE Transactions on Computers*, 39(4), 1990.

[15] S. J. Bavuso. Advanced reliability modeling of fault-tolerant computer-based systems. *NASA, TM-84501*, 1982.

[16] S.J. Bavuso. A User's View of CARE III. In *Reliability and Maintainability Symposium*, pages 382–389, 1984.

[17] S.J. Bavuso, J.B. Dugan, K.S. Trivedi, E.M. Rothmann, and W.E. Smith. Analysis of typical fault-tolerant architectures using HARP. *IEEE Transaction on Reliability*, 36:176–185, 1987.

[18] S.J. Bavuso, E. Rothmann, J.B. Dougan, K.S. Trivedi, N. Mittal, M.A. Boyd, R. M. Geist, and M.D. Smotherman Mark. HiRel: Hybrid Automated Reliability Predictor (HARP) Integrated Reliability Tool System. Technical report, 2003.

[19] M.D. Beaudry. Performance related reliability for computer systems. *IEEE Transactions on Computers*, 27(6), 1978.

[20] H. Beilner, J. Mäter, and N. Weissenberg. Towards a Performance Modelling Environment: News on Hit. In *Modelling Techniques and Tools for Computer Performance Evaluation*, 1989.

[21] S. Berson, E. de Souza, and R. Muntz. An object oriented methodology for the specification of markov models. *UCLA Technical Report CSD- 870030*, 1987.

[22] D. Bhandarkar. Analysis of memory interference in multiprocessors. *IEEE Transactions on Computers*, C-24:897–908, 1975.

[23] J. Bisson and R. Saint-German. The bs 7799 / iso 17799 standard for a better approach to information security. *Callio White Paper*, 2007.

[24] Hichem Boudali, Pepijn Crouzen, Boudewijn R. Haverkort, Matthias Kuntz, and Mariëlle Stoelinga. Arcade - a formal, extensible, model-based dependability evaluation framework. In *Proc. IEEE Int. Conf. on Engineering of Complex Computer Systems*, pages 243–248, 2008.

[25] D. Boudinova, F. Brecht, I. Mimikos, and A. Nowobilska. *CobiT at the Institue of Computer Science in Adlershof.* Final Project Report, Humboldt University Berlin, 2008.

[26] M. Bouissou. Boolean Logic Driven Markov Processes: A Powerful New Formalism for Specifying and Solving Very Large Markov Models. In *Proceedings of the PSAM6*, Puerto Rico, 2002.

[27] M. Bouissou. Automated Dependability Analysis of Complex Systems with the KB3 Workbench: the Experience of EDF R&D. In *Proceedings of the International Conference on ENERGY and ENVIRONMENT, CIEM 2005*, Bucharest, (Romania), 2005.

[28] M. Bouissou and J.L. Bon. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic Driven Markov Processes. *Reliability Engineering and System Safety*, 82(2):149–163, 2003.

[29] M. Bouissou, H. Bouhadana, M. Bannelier, and N. Villatte. Knowledge modelling and reliability processing: presentation of the FIGARO language and associated tools. In *Proceedings of the Safecomp'91*, Trondheim (Norway), 1991.

[30] M. Bouissou, Y Dituit, and S. Maillard. Reliability Analysis of a Dynamic Phased Mission System: Comparison of Two Approaches. *Modern Statistical and Mathematical Methods in Reliability*, pages 87–104, 2005.

[31] M. Bouissou and J.C. Houdebine. Inconsistency Detection in KB3 Models. In *Proceedings of the ESREL 2002*, Lyon, France, 2002.

[32] M. Bouissou, S. Humbert, S. Muffat, and N. Villatte. KB3 Tool: Feedback on Knowledge Bases. In *Proceedings of the ESREL 2002*, Lyon, France, 2002.

[33] M. Bouissou and Y. Lefebvre. A Path-Based Algorithm to Evaluate Asymptotic Unavailability for Large Markov Models. In *Proceedings of the RAMS 2002*, Seattle, USA, 2002.

[34] M. Bouissou and S. Muffat. High level representations for Markov analysis of complex dynamic systems. In *Proceedings of the IASTED Modeling and Simulation*, Marina del Rey, USA, 2004.

[35] C. Boulton. *Google Gmail, Google Apps Outage in the Cloud.* http://www.eweek.com/c/a/Messaging-and-Collaboration/Google-Gmail-Google-Apps-Suffer-Outage-in-The-Cloud/, 2008.

[36] E. Breton, M. Bouissou, and J. Aupied. A new tool for reliability studies of electrical networks with stand-by redundancies: OPALE. In *Proceedings of the PMAPS 2006*, 2006.

[37] J.J. Tifflerand L.A. Bryant and L. Guccione. CARE III final report phase I volume I and II. *NASA, Contractor Rep. 159122 and 159123*, 1979.

[38] BSI. Bsi-standard 100-3. *http://www.bsi.bund.de/literat/bsi_standard/standard_1003.pdf*, 2008.

[39] S. Burbeck. The Tao of e-business Services. *Emerging Technologies, IBM Software Group*, 2000.

[40] Rick Buskens and Oscar Gonzalez. *Model-Centric Development of Highly Available Software Systems*, chapter Model-Centric Development of Highly Available Software Systems, pages 163–187. Springer Verlag, 2007.

[41] R.W. Butler. An abstract language for specifying markov reliability models. *IEEE Transactions on Reliability*, 35, 1986.

[42] R.W. Butler. The sure reliability analysis program. *NASA TM 87593*, 1986.

[43] R.W. Butler and S.C. Johnson. Techniques for modeling the reliability of fault-tolerant systems with the markov state-space approach. *NASA Reference Publication 1348*, 1995.

[44] Hong Cai. A two steps method for analyzing dependency of business services on it services within a service life cycle. In *Proc. IEEE Int. Conf. on Web Services*, pages 877–884, 2006.

[45] P. Carer, J. Bellvis, M. Bouissou, J. Domergue, and J. Pestourie. A new method for Reliability assessment of electrical power supply with standby redundancies. In *Proceedings of the PMAPS*, 2003.

[46] J.A. Carrasco. Computationally efficient and numerically stable reliability bounds for repairable fault-tolerant systems. *IEEE Transactions on Computers*, 51(3), 2002.

[47] J.A. Carrasco. Computation of bounds for transient measures of large rewarded Markov models using regenerative randomization. *Computers and Operations Research*, 30(6), 2003.

[48] J.A. Carrasco. Solving dependability/performability irreducible Markov models using regenerative randomization. *IEEE Transactions on Reliability*, 52(3), 2003.

[49] J.A. Carrasco. Transient analysis of rewarded continuous time Markov models by regenerative randomization with Laplace transform inversion. *The Computer Journal*, 46(1), 2003.

[50] J.A. Carrasco. Solving large interval availability models using a model transformation approach. *Computers and Operations Research*, 31(5), 2004.

[51] J.A. Carrasco. Transient analysis of some rewarded Markov models using randomization with quasistationarity detection. *IEEE Transactions of Computers*, 53(9), 2004.

[52] J.A. Carrasco. Transient analysis of large Markov models with absorbing states using regenerative randomization. *Communications in StatisticsSimulation and Computation*, 34(4), 2005.

[53] J.A. Carrasco. Two methods to compute bounds for the distribution of cumulative reward for large Markov models. *Performance Evaluation*, 63(12), 2006.

[54] J.A. Carrasco and J. Figueras. METFAC: Design and Implementation of a Software Tool for Modeling and Evaluation of Complex Fault-Tolerant Computing Systems. In *Proceedings of the 16th FTCS*, 1993.

[55] J.A. Carrasco and V. Sune. METFAC User's Guide. http://dit.upc.es/qine/tools/metfac/guide.pdf, 2007.

[56] P.W. Chan, M.R. Lyu, and M. Malek. Reliable web services: Methodology, experiment and modeling. In *Proceedings International Conference on Web Services*, 2007.

[57] A. Charfi, B. Schmeling, and M. Mezini. Reliable messaging for bpel processes. In *Int. Conf. on Web Services*, pages 293–302, 2006.

[58] R.C. Cheung. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, 6, 1980.

[59] M. B. Chrissis, M. Konrad, and S. Shrum. *CMMI(R): Guidelines for Process Integration and Product Improvement*. Addison-Wesley Professional, 2006.

[60] G. Ciardo, A. Blakemore, P.F. Chimento, J.K. Muppala, and K.S. Trivedi. Automated generation and analysis of markov reward models using stochastic reward nets. *Linear Algebra, Markov Chains and Queuing Models*, 1992.

[61] G. Ciardo, J. K. Muppala, and K. S. Trivedi. Spnp: Stochastic petri net package. In *Proceedings of 3rd International Workshop on Petri Nets and Performance Models*, pages 142–150, 1989.

[62] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J.M. Doyle, W.H. Sanders, and R. Webster. The Möbius Modeling Tool. In *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, 2001.

[63] A.E. Conway and A. Goyal. Monte carlo simulation of computer system availability/reliability models. In *IBM Research Rep. RC 12459*, 1986.

[64] L.S. Crane, S. West, and A. Andrews. Analysis of octave support for hipaa security/privacy standards. *ATI IPT Technical Report 03-*, 2003.

[65] A da Silva, J. F. Martnez, L. Lopez, and L. Redondo. Exhaustif: A fault injection tool for distributed heterogeneous embedded systems. In *http://www.exhaustif.es/docs/exhaustif_article.pdf*, 2007.

[66] H. de Meer and H. Sevcikova. Xpenelope user guide, version 3.1. In *Technical Report, University Hamburg*, 1996.

[67] H. de Meer and H. Sevcikova. Penelope: dependability evaluation and the optimization of performability. In *In 9th Int. Conf. on Computer Performance Evaluation - Modelling Techniques and Tools*. Springer, 1997.

[68] E. de Souza e Silva, H.R. Gail, and R.V. Campos. Calculating transient distributions of cumulative reward. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 231–240, 1995.

[69] L.J. Dolny, R.E Fleming, and R.L. De Hoff. Fault-tolerant computer system design using GRAMP. In *Proceedings of the 1983 Annual Reliability and Maintainability Symposium*, pages 417–422, 1983.

[70] J.B. Dugan, K.S. Trivedi, R. Geist, and V. Nicola. Extended stochastic Petri nets: Applications and analysis. In *Proceedings of the Models of Computer System Performance*, 1985.

[71] Eclipse. Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/, 2008.

[72] Eclipse. Graphical Modeling Framework. http://www.eclipse.org/modeling/gmf/, 2008.

[73] Eclipse. M2M Project. http://www.eclipse.org/m2m/, 2008.

[74] Eclipse. Rich Client Platform. http://www.eclipse.org/home/categories/rcp.php, 2008.

[75] E. Elsayed. *Reliability Engineering*. Addison Wesley Longman, 1996.

[76] V. Ermagan, I. Kruger, and M. Menarini. A fault tolerance approach for enterprise applications. In *IEEE International Conference on Services Computing*, 2008.

[77] Abdelkarim Erradi and Piyush Maheshwari. A broker-based approach for improving web services reliability. In *Proceedings of the IEEE International Conference on Web Services*, pages 355–362, 2005.

[78] C. Béounes et al. Surf-2: A program for dependability evaluation of complex hardware and software systems. In *23rd Int. Symp. on Fault-Tolerant Computing*, pages 668–673, 1993.

[79] S.J. Farlow. *An Introduction to Differential Equations and Their Applications*. Dover Pubn Inc, 2006.

[80] R. E. Fleming. *Coherent system repair models*. Ph.D. dissertation, T.R. 195, Dept. of Operations Research and Dept. of Statistics, Stanford Univ., Stanford, Calif., 1980.

[81] R.E Fleming and L.J. Dolny. Fault-tolerant design-to-specs with GRAMP & GRAMS. In *Proceedings of the 1984 Annual Reliability and Maintainability Symposium*, pages 403–408, 1984.

[82] J. Fussell. How to calculate system reliability and safety characteristics. *IEEE Transact. Reliab.*, 24(3):169–174, 1975.

[83] B. P. Gallagher, M. Phillips, K. J. Richter, and S. Shrum. *CMMI-ACQ: Guidelines for Improving the Acquisition of Products and Services*. Addison-Wesley Professional, 2009.

[84] M. Gallois and M. Pillire. Benefits Expected from Automatic Studies with KB2 in PSAs at EDF. In *Proceedings of the PSA99*, Washington, USA, 1999.

[85] R. Geist and K. S. K.S. Trivedi. Ultrahigh reliability prediction for fault-tolerant computer systems. *IEEE Transactions on Computers*, 12, 1983.

[86] R. Geist and K. S. Trivedi. Ultrahigh reliability prediction for fault-tolerant computer systems . *IEEE Transactions on Computers*, 12, 1983.

[87] S. Gokhale, W.E. Wong, K. Trivedi, and J.R. Horgan. An analytical approach to architecture based software reliability prediction. In *Proceedings of the 3rd International Computer Performance and Dependability Symposuim*, 1998.

[88] T. Goldschmidt. *Entwicklung eines Modells für die Verfügbarkeitsbewertung auf Basis generischer Referenzmodelle (Eng.: Development of an availability assessment model based on generic reference models)*. Master Thesis, University of Magdeburg, 2009.

[89] L. Gönczy, S. Chiaradonna, F. Di Giandomenico, A. Pataricza, A. Bondavalli, and T. Bartha. Dependability evaluation of web service-based processes. In M. Telek, editor, *in Proceedings of European Performance Engineering Workshop (EPEW 2006)*, Lecture Notes on Computer Science, pages 166–180, Budapest, HUNGARY, 2006. Springer.

[90] A. Goyal, W.C. Carter, E.S. de Souza, S.S. Lavenberg, and K.S. Trivedi. The system availability estimator. In *Proc. IEEE 16th Annual Symposium on Fault-Tolerant Computing*, pages 84–89, 1986.

[91] A. Goyal, S.S. Lavenberg, and K.S. Trivedi. Probabilistic modeling of computer system availability. In *IBM Research Rep. RC 11076*, 1985.

[92] A. Goyal, S.S.Lavenberg, and K.S.Trivedi. Probabilistic modeling of computer system availability. *Annals of Operations Research*, 8, 1987.

[93] M. Grottke, H. Sun, R.M. Fricks, and K.S. Trivedi. Ten Fallacies of Availability and Reliability Analysis. In *Proceedings of the 5th International Service Availability Symposium (ISAS), Tokyo, Japan*, pages 187–206. Springer Verlag, 2008.

[94] W3C Working Group. Web Services Architecture. *http://www.w3.org/TR/ws-arch/*, 2004.

[95] Huipeng Guo, Jinpeng Huai, Huan Li, Ting Deng, Yang Li, and Zongxia Du. Angel: Optimal configuration for high available service composition. In *IEEE International Conference on Web Services*, pages 280–287, 2007.

[96] Riadh Ben Halima, Khalil Drira, and Mohamed Jmaiel. A qos-oriented reconfigurable middleware for self-healing web services. In *Proc. 2008 IEEE International Conference on Web Services*, pages 104–111, 2008.

[97] Andreas Hanemann, David Schmitz, and Martin Sailer. A framework for failure impact analysis and recovery with respect to service level agreements. In *Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 49–58, 2005.

[98] B. R. Haverkort and I. G. Niemegeers. Performability modeling tools and techniques. In *University of Twente, TeleInformatics and Open Systems*, 1996.

[99] B. R. Haverkort and I. G. Niemegeers. *Performability Modelling Tools and Techniques*. University of Twente, 1996.

[100] B. R. Haverkort, I. G. Niemegeers, and P. Veldhuyzen van Zanten. DyQNtool A Performability Modeling Tool Based on the Dynamic Queuing Network Concept. *Computer Performance Evaluation: Modeling Techniques and Tools*, 1992.

[101] B.R. Haverkort and I.G. Niemegeers. Performability modeling tools and techniques. *University of Twente, Tele-Informatics and Open Systems*, 1996.

[102] B.R. Haverkort and K.S. Trivedi. Specification and generation of markov reward models. *Discrete-Event Dynamic Systems: Theory and Applications*, 3:219–247, 1993.

[103] C. Hirel, R. Sahner, X. Zang, and K. Trivedi. Reliability and performability modeling using sharpe 2000. In *Technical Report Center for Advanced Computing and Communication Department of Electrical and Computer Engineering Duke University, Durham*, 2000.

[104] W. Hoarau, S. Tixeuil, , and F. Vauchelles. *Easy Fault Injection and Stress Testing with FAIL-FCI.* LRI-CNRS 8623 et INRIA Grand Large, 2005.

[105] IT Governance Institute. *CobiT 4.1.* http://www.isaca.org/Content/NavigationMenu/Members_and_Leaders/COBIT6/Obtain_COBIT/Obtain_COBIT.htm, 2009.

[106] IsoGraph. Attack Tree Technical Specification. http://www.isograph-software.com/_techspecs/attacktree+V1TS.pdf, 2008.

[107] IsoGraph. AvSim+ Technical Specification. http://www.isographsoftware.com/_techspecs/avsim32techspec.pdf, 2008.

[108] IsoGraph. FaultTree+ Technical Specification. http://www.isographsoftware.com/_techspecs/psa32techspec.pdf, 2008.

[109] IsoGraph. Network Availability Program Technical Specification. http://www.isograph-software.com/_techspecs/nap32techspec.pdf, 2008.

[110] IsoGraph. Reliability Workbench Technical Specification. http://www.isograph-software.com/_techspecs/wk32techspec.pdf, 2008.

[111] Z. Jelinski and P.B. Moranda. Software Reliability Research. *Statistical Computer Performance Evaluation*, 1972.

[112] A. M. Johnson and Malek M. Survey of software tools for evaluating reliability, availability, serviceability. *ACM Computing Surveys*, pages 227–269, 1988.

[113] Mohamed Kaâniche, Karama Kanoun, and Magnos Martinello. A user-perceived availability evaluation of a web based travel agency. In *Proceedings of 2003 International Conference on Dependable Systems and Networks*, pages 709–716, 2003.

[114] K. Kanoun and M. Borrel. Dependability of fault-tolerant systems explicit modeling of the interactions between hardware and software components. In *IEEE International Computer Performance and Dependability Symposium*, 1996.

[115] K. Kanoun, M. Borrel, T. Moreteveille, and A. Peytavin. Modeling the dependability of cautra, a subset of the french air traffic control system. In *26th Int. Symp. Fault-Tolerant Computing*, pages 495–515, 1996.

[116] Barry Kirwan. *A Guide to Practical Human Reliability Assessment.* Taylor and Francis Ltd., London, 1994.

[117] J. Klensin. Simple mail transfer protocol. *RFC 2821*, 2001.

[118] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series).* Prentice Hall PTR, 2004.

[119] S. Krishnamurthy and A.P. Mathur. On the estimation of reliability of a software system using reliabilities of its components. In *Proceedings of the 8th International Symposium on Software Reliability*, 1997.

[120] P. Kubat. Assessing reliability of modular software. *Operation Research Letters*, 8, 1989.

[121] K. Kukreja and D. Jasso. An insider's view to the HP Universal CMDB. *HP Technical White Paper*, 2008.

[122] J.H. Lala. Interactive reductions in the number of states in Markov reliability analysis. In *Proceedings of the AZAA Guidance and Controls Conference*, 1983.

[123] J.H. Lala. *Mark1 Markov modeling package.* The Charles Stark Draper Laboratory, Cambridge, Mass., 1983.

[124] J.C. Laprie. Dependability evaluation of software systems in operation. *IEEE Transactions on Software Engineering*, 10, 1984.

[125] J. Ledoux. Availability modeling of modular software. *IEEE Transactions on Reliability*, 48, 1999.

[126] D. Lee, J. Abraham, D. Rennels, and G. Gilley. A numerical technique for the evaluation of large, closed fault-tolerant systems. *Dependable Computing for Critical Applications*, 1992.

[127] R. Lepold. *PENPET: A Performability Modeling Evaluation Tool Based on Stochastic Petri Nets.* 1991.

[128] Qianhui Althea Liang, Herman Lam, Lalita Narupiyakul, and Patrick C. K. Hung. A rule-based approach for availability of web service. In *Proc. 2008 IEEE International Conference on Web Services*, pages 153–160, 2008.

[129] LINA & INRIA Atlas Group. ATL: Atlas Transformation Language User Manual. ttp://www.eclipse.org/m2m/atl/doc/ATL User Manual[v0.7].pdf., 2006.

[130] B. Littlewood. Theories of software reliability: how good are they and how can they be improved? *IEEE Transactions on Software Engineering*, 6, 1980.

[131] J. Losq. Effects of Failures on Gracefully Degradable Systems. In *Proceedings of the International Symposiym on Fault-Tolerant Computing*, 1977.

[132] M.R. Lyu. *Handbook of Software Reliability Engineering.* McGraw-Hill, 1995.

[133] M.R. Lyu and J. Schoenwaelder. A Web-Based Tool for Software Reliability Measurement. In *Proceedings of International Symposium on Software Reliability Engineering*, pages 382–389, Padeborn, Germany, 1998.

[134] I. Majzik, P. Domokos, and M. Magyar. Tool-supported dependability evaluation of redundant architectures in computer based control systems. In E. Schnieder and G. Tarnai, editors, *Proc. FORMS/FORMAT 2007, the 6th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems, 25-26 January 2007, Braunschweig, Germany*, pages 342–352. GZVB, Braunschweig, Germany, 2007.

[135] S.V. Makam and A. Avizienis. ARIES 81: A reliability and life-cycle evaluation tool for fault-tolerant systems. *Digest of the 12th Annual Symposium on Fault-Tolerant Computing*, pages 267–274, 1981.

[136] S.V. Makam, A. Avizienis, and G. Grusas. UCLA ARIES 82 users guide. *Tech. Rep. CSD- 82030, Computer Science Dept., Univ. of California, Los Angeles.*, 1982.

[137] M. Malek, B. Milic, and N. Milanovic. Analytical availability assessment of it services. In *Service Availability: 5th International Service Availability Symposium, ISAS 2008 Proceedings*, volume 5017 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2008.

[138] Miroslaw Malek, Guenther Hoffmann, Nikola Milanovic, Stefan Bruening, Reinhard Meyer, and Bratislav Milic. Methoden und Werkzeuge zur Verfgbarkeitsermittlung. Technical Report 219, Humboldt University Berlin, 2007.

[139] 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*, 27&28, 1996.

[140] A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modeling with Generalized Stochastic Petri Nets.* Wiley, 1995.

[141] A. Marsan, M. Balbo, and G. Conte. A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(1), 1984.

[142] H. Mauser. Implementierung eines optimierungsverfahrens für rekonfigurierbare systeme. In *Diploma Thesis, University Erlangen-Nürnberg*, 1990.

[143] J.F. Meyer. On Evaluating the Performability of Degradable Computing Systems. *IEEE Transactions of Computers*, 28(8), 1980.

[144] Microsoft. *Microsoft Operations Framework 4.0.* http://technet.microsoft.com/en-gb/library/cc506049.aspx, 2008.

[145] N. Milanovic. Contract-based Web Service Composition Framework with Correctness Guarantees. In *Proceedings of the 2nd International Service Availability Forum (ISAS)*, pages 46–59, Berlin, Germany, 2005.

[146] N. Milanovic. *Contract-based Web Service Composition.* PhD Dissertation, Humboldt University Berlin, 2006.

[147] N. Milanovic, B. Milic, and M. Malek. Modeling business process availability. In *SERVICES '08: Proceedings of the 2008 IEEE Congress on Services - Part I*, pages 315–321, Washington, DC, USA, 2008. IEEE Computer Society.

[148] K.B. Misra. *New Trends in System Reliability Evaluation.* Elsevier, 1993.

[149] M.K. Molloy. *On The Integration of Delay and Throughput Measures in Distributed Processing Models.* PhD Dissertation, UCLA, 1981.

[150] A. Movaghar and J.F. Meyer. Performability modeling with stochastic activiy networks. In *Proceedings of the Real-Time Systems Symposium*, 1984.

[151] B. Mueller. NUMAS: A Tool for the Numerical Modelling of Computer Systems. In *Proc. International Conference on Modelling Techniques and Tools for Performance Analysis*, 1984.

[152] J.D. Musa, A. Iannino, and K. Okumoto. *Software Reliability - Measurement, Prediction, Application.* McGraw-Hill, 1987.

[153] A.M. Neufelder. How to measure the impact of specific development practices on fielded defect density. In *Proceedings. 11th International Symposium on Software Reliability Engineering (ISSRE)*, 2000.

[154] A.M. Neufelder. How to predict software defect density during proposal phase. In *Proceedings of the IEEE National Aerospace and Electronics Conference (NAECON)*, 2000.

[155] Y.W. Ng and A. Avizienis. ARIES-an automated reliability estimation system. In *Proceedings of the 1977 Annual Reliability and Maintainability Symposium*, pages 108–113, 1977.

[156] OMG. Meta Object Facility (MOF) 2.0 Core Specification. http://www.omg.org/ cgi-bin/apps/doc?ptc/03-10-04.pdf., 2004.

[157] Object Management Group (OMG). Business Process Modeling Notation Specification. *http://www.bpmn.org/Documents/OMG Final Adopted BPMN 1-0 Spec 06-02-01.pdf*, 2006.

[158] T.W. Page, S.E. Berson, W.C. Cheng, and R.R. Muntz. An object oriented modeling environment. In *Proceedings of Conference on Object-oriented programming systems, languages and applications*, 1989.

[159] A. Pataricza, I. Majzik, G. Huszerl, and Gy. Várnai. UML-based design and formal analysis of a safety-critical railway control software module. In G. Tarnai and E. Schnieder, editors, *Formal Methods for Railway Operation and Control Systems (Proceedings of Symposium FORMS-2003, Budapest, Hungary, May 15-16)*, pages 125–132. L' Harmattan, Budapest, 2003.

[160] D. Patterson. A simple way to estimate the cost of downtime. In *Proceedings of the 16th System Administrator Conference - LISA'02*, 2002.

[161] M.C. Paulk. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Addison-Wesley Longman, 1994.

[162] J.C. Perez. *Extended Gmail outage hits Apps admins.* http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9117322, 2008.

[163] J.C. Perez. *Google Apps Customers Miffed Over Downtime.* http://www.pcworld.com/businesscenter/article/130234/google_apps_customers_miffe, 2008.

[164] E. Pinheiro, W.-D. Weber, and Luiz André Barroso. Failure Trends in a Large Disk Drive Population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, page 1728, 2008.

[165] A. Polze, N. Milanovic, and M. Schoebel. *Fundamentals of Service-oriented Engineering.* HPI, University of Potsdam, 2006.

[166] J. Pukite and P. Pukite. *Modeling for Reliability Analysis: Markov Modeling for Reliability, Maintainability, Safety, and Supportability Analyses of Complex Systems.* IEEE Press, 1998.

[167] A. Puliafito, O. Tomarchio, and L. Vita. Porting sharpe on the web: Design and implementation of a network computing platform using java. *Computer Performance Evaluation*, 1997.

[168] S. Rai, Veeraraghavan M., and K. Trivedi. A Survey of Efficient Reliability Computation Using Disjoint Products Approach. *Networks*, 25, 1995.

[169] Florian Rosenberg, Christian Platzer, and Schahram Dustdar. Bootstrapping performance and dependability attributes of web services. In *Proc. IEEE International Conference on Web Services*, pages 205–212, 2006.

[170] R. Sahner, K.S. Trivedi, and A. Puliafito. Sharpe 2000 tool manual. In *http://www.ee.duke.edu/˜chirel/MANUAL/manualSharpe.pdf*, 2000.

[171] R. A. Sahner and K.S. Trivedi. A hierarchical, combinatorial-markov method of solving complex reliability models. In *Proceedings of the 1986 Full Joint Computer Conference*, pages 817–825, 1986.

[172] R. A. Sahner and K.S. Trivedi. Reliability modeling using sharpe. *IEEE Trans. Reliability*, 36(2), 1987.

[173] R.A. Sahner, K.S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems*. Kluwer Academic Publishers, 2002.

[174] W.H. Sanders. Möbius Manual. http://www.perform.csl.uiuc.edu/mobius/manual/ MobiusManual.pdf, 2006.

[175] W.H. Sanders and J.F. Meyer. METASAN: a performability evaluation tool based on stochastic activity networks. In *Proceedings of 1986 ACM Fall joint computer conference*, 1986.

[176] W.H. Sanders and J.F. Meyer. Stochastic activity networks: formal definitions and concepts. *Lectures on formal methods and performance analysis: first EEF/Euro summer school on trends in computer science*, pages 315–343, 2002.

[177] N. Sato and K. S. Trivedi. Stochastic modeling of composite web services for closed-form analysis of their performance and reliability bottlenecks. In *Proceedings of the 5th international conference on Service-Oriented Computing*, pages 107–118, 2007.

[178] N. Sato and K.S. Trivedi. Accurate and efficient stochastic reliability analysis of composite services using their compact markov reward model representations. In *Proceedings of the IEEE International Conference on Services Computing, SCC 2007*, 2007.

[179] Gary W. Scheer and David J. Dolezilek. Comparing the Reliability of Ethernet Network Topologies in Substation control and Monitoring Networks. *Schweitzer Engineering Laboratories Technical Report 6103*, 2004.

[180] A.Q. Scheuing, K. Frühauf, and W. Schwarz. Maturity Model for IT Operations (MITO). In *2nd World Congress on Software Quality*, 2000.

[181] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, pages 1–16, 2008.

[182] B. P. Shah. *Analytic solution of stochastic activity networks with exponential and deterministic activities*. 1993.

[183] Lingshuang Shao, Lu Zhang, Tao Xie, Junfeng Zhao, Bing Xie, and Hong Mei. Dynamic availability estimation for service selection based on status identification. In *Proceedings of the 2008 IEEE International Conference on Web Services*, pages 645–652, 2008.

[184] T.C. Sharma and I. Bazovsky. Reliability Analysis of Large System By Markov Techniques. In *Proceedings of the IEEE Annual Reliability and Maintainability Symposium*, 1993.

[185] M.L. Shooman. Computer Software Reliability: Many-State Markov Modeling Techniques. *RADC-TR-75-169, Rome Air Development Center*, 1975.

[186] D. Siewiorek and R. Swarz. *The Theory and Practice of Reliable System Design*. Digital Press, 1982.

[187] T.B. Smith and J.H. Lala. Developmentand evaluation of a fault-tolerant multiprocessor (FTMP) computer. NASA-CR-172286, 1986.

[188] W. E. Smith, K. S. Trivedi, L. Tomek, and J. Ackeret. Availability analysis of multi-component blade server systems. *IBM Systems Journal*, 2008.

[189] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. 1994.

[190] D. Stott, P.H. Jones, M. Hamman, Z. Kalbarczyk, and R.K. Iyer. NFTAPE: Networked Fault Tolerance and Performance Evaluator. In *Proceedings of the International Conference on Dependable Systems and Networks*, 2002.

[191] D.T. Stott. *Automated Fault Injection Based Dependability Analysis of Distributed Computer Systems*. PhD Thesis, University of Illinois, 2000.

[192] D.T. Stott, B. Floering, Z. Kalbarczyk, and R.K. Iyer. Dependability Assessment in Distributed Systems with Lightweight Fault Injectors in NFTAPE. In *Proceedings of the 4th International Computer Performance and Dependability Symposuim*, pages 91–100, 2000.

[193] A.T. Tai, J.F. Meyer, and A. Avizienis. Software Performability: From Concepts to Applications. *International Series in Engineering and Computer Science*, 347, 1996.

[194] Dong Tang, Dileep Kumar, Sreeram Duvur, and Oystein Torbjornsen. Availability measurement and modeling for an application server. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 669, 2004.

[195] Dong Tang, Ji Zhu, and Roy Andrada. Automatic generation of availability models in rascad. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 488–494, 2002.

[196] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy. Coordinated forward error recovery for composite web services. In *Proceeding of the 22nd International Symposium on Reliable Dependable Systems, SRDS 2003*, pages 167–176, Florence, Italy, 2003.

[197] Amazon EC-2 Support Team. Amazon EC-2 Outage Report. http://developer. amazonwebservices.com/connect/message.jspa?messageID=56849 # 56849, 2008.

[198] CMMI Product Team. Cmmi for systems engineering and software engineering. *Software Engineering Institute, Carnegie Mellon, http://www.sei.cmu.edu/cmmi/*, 2009.

[199] S. Tixeuil, W. Hoarau, and L. Silva. *An Overview of Existing Tools for Fault-Injection and Dependability Benchmarking in Grids*. CoreGRID Technical Report, 2006.

[200] S. Tixeuil, W. Hoarau, and L. Silva. An overview of existing tools for fault-injection and dependability benchmarking in grids. In *CoreGRID Technical Report*, 2006.

[201] K.S. Trivedi. SHARPE 2000 GUI Manual. In *http://www.ee.duke.edu/˜chirel/ MANUAL/gui.pdf*, 2000.

[202] K.S. Trivedi. Harpe 2002: Symbolic hierarchical automated reliability and performance evaluator. In *Proceedings International Conference on Dependable Systems and Networks (DSN)*, 2002.

[203] K.S. Trivedi, G. Ciardo, M. Malhotra, and R.A. Sahner. Dependability and performability analysis. *Technical Report ICASE 93-85, NASA Langley Research Center*, 1993.

[204] K.S. Trivedi, J.K. Muppala, S.P. Woolet, and B.R. Haverkort. Composite performance and dependability analysis. *Performance Evaluation*, 14(3&4), 1992.

[205] W. Pitt Turner, J. Seader, and K. Brill. Industry Standard Tier Classification Define Site Infrastructure Performance. White paper, The Uptime Institute, 2005.

[206] J. Tvedt. *Solution of large-sparse stochastic process representations of stochastic activity networks*. 1990.

[207] International Telecommunications Union. Final draft of revised Recommendation E.800. http://www.itu.int/md/T05-SG02-080506-TD-WP2-0121/en, 2008.

[208] W. van der Aalst, A. Hofstede, and M. Weske. Business Process Management: A Survey. In *International Conference on Business Process Management (BPM 2003)*, pages 1–12. Springer Verlag, 2003.

[209] H. van Loon. *Process Assessment and ISO/IEC 15504: A Reference Book*. Springer, 2004.

[210] A. P. A. van Moorsel and W. H. Sanders. Adaptive uniformization. *ORSA Communications in Statistics: Stochastic Models*, 10(3), 1994.

[211] A. P. A. van Moorsel and K. Wolter. Analysis of restart mechanisms in software systems. *IEEE Transactions on Software Engineering*, 32(8), 2006.

[212] M. Vieira and N. Laranjeiro. Comparing web services performance and recovery in the presence of faults. In *IEEE International Conference on Web Services*, pages 623–630, 2007.

[213] Marco Vieira and Henrique Madeira. Recovery and performance balance of a cots dbms in the presence of operator faults. In *Proc. Int. Conf. on Dependable Systems and Networks*, pages 615–626, 2002.

[214] W. Vogels. Web Services are not Distributed Objects: Common Misconceptions about the Fundamentals of Web Service Technology. *IEEE Internet Computing*, 7(6):59–66, 2003.

[215] M. Walter, M. Siegle, and A. Bode. OpenSESAME - The simple but extensive, structured availability modeling environment. In *Reliability Engineering and System Safety*, 2007.

[216] L. Wells. Performance analysis using CPN tools. In *Proceedings of the 1st International Conference on Performance Evaluation Methodologies and Tools*, 2006.

[217] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, 2007.

[218] S. West and A. Andrews. Octave-best practices comparative analysis. *ATI IPT Technical Report 03-4*, 2003.

[219] Guoquan Wu, Jun Wei, Xiaoqiang Qiao, and Lei Li. A bayesian network based qos assessment model for web services. In *IEEE International Conference on Services Computing*, pages 498–505, 2007.

[220] S. Yacoub, B. Cukic, and H. Ammar. Scenario-based reliability analysis of component-based software. In *Proceedings of the 10th International Symposium on Software Reliability*, 1999.

[221] M. Zaddach. Wege der transienten optimierung ein zeitdiskreter ansatz fr penelope. In *Diploma Thesis, University Hamburg*, 1998.

[222] L. Zeng, B. Benatallah, M. Dumas, and J. Kalagnanam. Quality Driven Web Services Composition. In *Proceedings of the 12th International Conference World Wide Web*, pages 411 – 421, Budapest, Hungary, 2003.

[223] L. Zeng, B. Benatallah, A.H.H Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions of Software Engineering*, 30(5):311–327, 2004.

[224] Jia Zhang and Liang-Jie Zhang. Criteria analysis and validation of the reliability of web services-oriented systems. In *Proceedings of the IEEE International Conference on Web Services*, pages 621–628, 2005.

[225] Wenbing Zhao. Byzantine fault tolerant coordination for web services atomic transactions. In *Proceedings of the 5th international conference on Service-Oriented Computing*, pages 307–318, 2007.

[226] Wenbing Zhao and Honglei Zhang. Byzantine fault tolerant coordination for web services business activities. In *Proceedings of the 2008 IEEE International Conference on Services Computing*, pages 407–414, 2008.

[227] Zibin Zheng and M.R. Lyu. A distributed replication strategy evaluation and selection framework for fault tolerant web services. In *IEEE International Conference on Web Services*, pages 145–152, 2008.

Appendix A

Möbius Availability Model

The model of error-handling module with associated case probabilities and gate functions is given in Figure A.1 and Tables A.1 and A.2 respectively. The model of the I/O module with associated case probabilities and gate functions is given in Figure A.2 and Tables A.3 and A.4 respectively. Finally, the model of the memory with associated case probabilities and gate functions is given in Figure A.3 and Tables A.5, A.6 and A.7 respectively.

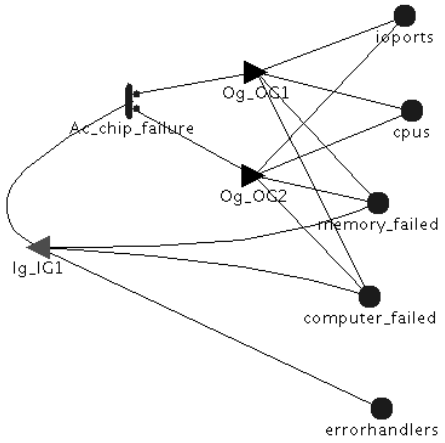

Figure A.1: SAN model of the error-handling unit

Case	Probability
1	`return(comp_cov);`
2	`return(1.0-comp_cov);`

Table A.1: Case probabilities for *Ac_chip_failure* activity from the Figure A.1

Gate	Function

Og_OG1	`cpus->Mark()=0;`
	`ioports->Mark()=0;`
	`memory_failed->Mark()=2;`
	`computer_failed->Mark()++;`

Og_OG2	`cpus->Mark()=0;`
	`ioports->Mark()=0;`
	`errorhandlers->Mark()=0;`
	`memory_failed->Mark()=2;`
	`computer_failed->Mark()=num_comp;`

IG_IG1	`errorhandlers->Mark()==2) &&`
	`(memory_failed->Mark()<2) &&`
	`computer_failed->Mark() < num_comp)`

Table A.2: Gate functions for the error-handling module from Figure A.1

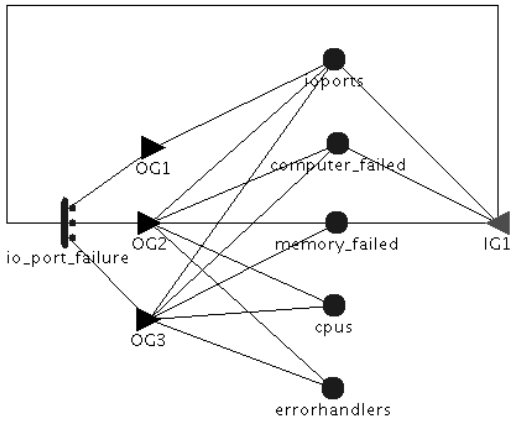

Figure A.2: SAN model of the I/O port

Case	Probability

1	`if(ioports->Mark() == 2)`
	`return(IO_cov);`
	`else`
	`return(0.0);`

2	`if(ioports->Mark() == 2)`
	`return(1.0-IO_cov)*comp_cov);`
	`else`
	`return(comp_cov);`

3	`if(ioports->Mark() == 2)`
	`return((1.0-IO_cov)*(1.0-comp_cov));`
	`else`
	`return(1.0-comp_cov);`

Table A.3: Case probabilities for *io_port_failure* activity from the Figure A.2

Gate	Function
OG1	`if(ioports->Mark()==2)` `  ioports->Mark()--;`

OG2	`cpus->Mark()=0;`
	`ioports->Mark()=0;`
	`errorhandlers->Mark()=0;`
	`memory_failed->Mark()=2;`
	`computer_failed->Mark()++;`

OG3	`cpus->Mark()=0;`
	`ioports->Mark()=0;`
	`errorhandlers->Mark()=0;`
	`memory_failed->Mark()=2;`
	`computer_failed->Mark()=num_comp;`

IG1	`(ioports->Mark()>0) &&`
	`(memory_failed->Mark()<2) &&`
	`computer_failed->Mark()<num_comp)`

Table A.4: Gate functions for the I/O module from Figure A.2

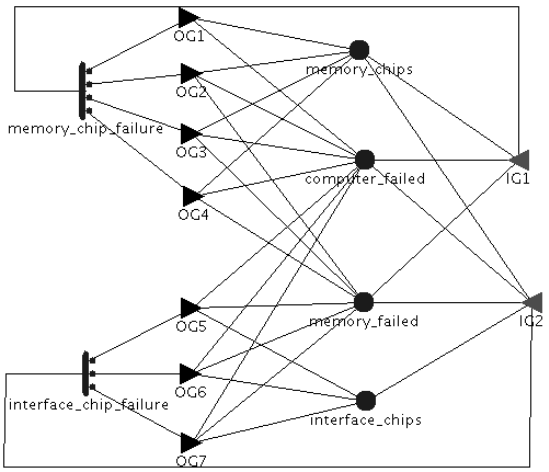

Figure A.3: SAN model of the memory module

Case	Probability

1	`if(memory_chips->Mark() == 39)`
	`return(0.0);`
	`else`
	`return(RAM_cov);`

2	`if(memory_chips->Mark() == 39)`
	`return(mem_cov);`
	`else`
	`return((1.0-RAM_cov)*mem_cov);`

3	`if(memory_chips->Mark() == 39)`
	`return((1.0-mem_cov)*comp_cov);`
	`else`
	`return((1.0-RAM_cov)*(1.0-mem_cov)*comp_cov);`

4	`if(memory_chips->Mark() == 39)`
	`return((1.0-mem_cov)*(1.0-comp_cov));`
	`else`
	`return((1.0-RAM_cov)*(1.0-mem_cov)*(1.0-comp_cov));`

Table A.5: Case probabilities for *memory_chip_failure* activity from the Figure A.3

Case	Probability
1	`return(mem_cov);`
2	`return ((1.0-mem_cov)*comp_cov);`
3	`return ((1.0-mem_cov)*(1.0-comp_cov));`

Table A.6: Case probabilities for *interface_chip_failure* activity from the Figure A.3

Gate	Function
OG1	`if(memory_chips->Mark()>39)` `memory_chips->Mark()--;`

OG2	`memory_chips->Mark()=0;`
	`interface_chips->Mark()=0;`
	`memory_failed->Mark()++;`
	`if(memory_failed->Mark()>1)`
	`computer_failed->Mark()++;`

OG3	`memory_chips->Mark()=0;`
	`interface_chips->Mark()=0;`
	`if((memory_failed->Mark()==1) && (computer_failed->Mark()<=num_comp-2))`
	`computer_failed->Mark()=num_comp;`
	`else`
	`computer_failed->Mark()++;`
	`memory_failed->Mark()=2;`

OG4	`memory_chips->Mark()=0;`
	`interface_chips->Mark()=0;`
	`memory_failed->Mark()=2;`
	`computer_failed->Mark()=num_comp;`

OG5	`interface_chips->Mark()=0;`
	`memory_failed->Mark()++;`
	`if(memory_failed->Mark()>1)`
	`computer_failed->Mark()++;`

OG6	`interface_chips->Mark()=0;`
	`if((memory_failed->Mark()==1) && (computer_failed->Mark()<=num_comp-2))`
	`computer_failed->Mark()=num_comp;`
	`else`
	`computer_failed->Mark()++;`
	`memory_failed->Mark()=2;`

OG7	`interface_chips->Mark()=0;`
	`memory_failed->Mark()=2;`
	`computer_failed->Mark()=num_comp;`

IG1	`(memory_chips->Mark()>38) && (computer_failed->Mark()<num_comp) &&` `(memory_failed->Mark()<2)`
IG2	`(interface_chips->Mark()>1) && (memory_failed->Mark()<2) &&` `(computer_failed->Mark()<num_comp)`

Table A.7: Gate functions for the memory module from Figure A.3

Appendix B

Availability Assessment Tools

B.1 General Purpose Quantitative Modeling Tools

B.1.1 ACARA

1. Source/Reference: N/A.

   Web: `http://www.openchannelfoundation.org/projects/ACARA_II`

2. Project status

   Acara is under development by the NASA Glen Research Center in Cleveland. It exists in the current version under the name ACARA-2.

3. License type: Open-Channel-Foundation, California Institute of Technology.

4. General purpose

   ACARA (Availability Cost and Resource Allocation) is used for the reliability, lifecycle costs and resource planing analysis of the periodical repairable systems. It combines the usage of exponential and Weibull distributions to simulate the usage duration of each system component. The replacement of each failed component is simulated to optimize system performance and to discover the constraints of the available resources and developed components. The constraints are user defined. ACARA evaluates the system availability using block diagrams.

5. Platform: Windows.

6. Model

   (a) Model class: quantitative, simulation; IT-level.

   (b) Model type: Monte-Carlo methods, Weibull and exponential distributions, block diagrams, models for early and random failures.

   (c) Model description
   ACARA models the following events:

- Time to failure: For each block in a diagram the time to failure is calculated using exponential and Weibull distributions and failure models described below. The models are parameterized with user-specific data, which approximate failure properties of each block.
- Down time: ACARA estimates the down time of each failed block, until it is replaced. In case where replacement block is locally available, it is immediately applied and downtime depends only on MTTR. If the replacement block is not locally available, ACARA plans the replacement procedure using production and delivery capacities of the replacement part. The tool also checks the conditions under which a failed block may be replaced.
- Time to repair: When all conditions that are required for a block replacement are fulfilled, ACARA estimates the time to replace the failed block. Time to repair depends on the MTTR of the observed block. Three additional factors are also taken into account: repair team, equipment and automatization.

(d) Systems modeled: repairable systems.

(e) Model input

MTTR of each block is given in a tabular form (Repair Time and Personnel Quantities Input Table). Further inputs are user-defined values for simulation parameters (simulation and replacement duration, fault periods) and cost parameters (salaries, transport, equipment, robots).

(f) Model output

The tool generates four types of output values:

- Performability: time-capacity diagram, state availability, overall system availability, equivalent availability, cumulative availability, reliability, continuous state behavior.
- Failure and repair: failure density function, cumulative density function, failure occurrence, early and random failures, repair cumulative density function, repair ability, criticality.
- Lifecycle costs: hardware, transport, personnel, equipment, robots, total costs.
- Resource allocation: storage status, delivered hardware, resource usage.

(g) Interfaces

- Input: manual terminal input.
- Output: graphical or textual files.

7. Use cases

   The tool is developed and used by NASA.

8. Assumptions and restrictions

   All repair tasks are performed simultaneously.

B.1.2 ARIES

1. Source/Reference: [135], [136], [155], [14]

2. Project status

   Aries was developed in 1977 [135], followed by versions ARIES 81 [136] and ARIES 82 [155]. Extensions have been proposed in 1990 in [14], it is unknown whether they have been implemented.

3. License type: Last known implementation is from 1982. Current license type is unknown.

4. General purpose

   Reliability and life-cycle analysis for fault-tolerant systems.

5. Platform: APL and C implementation.

6. Model

   (a) Model class: analytical, IT level.

   (b) Model type: Homogeneous Markov model, solved by Lagrange-Sylvester interpolation.

   (c) Model description

   The system can be specified using a state transition matrix or as a series of independent subsystems each containing identical modules that are either active or serve as spares. It uses a matrix transformation solution technique that assumes distinct eigenvalues for the state transition matrix.

   (d) Systems modeled: (closed) nonrepairable, repairable, with transient fault recovery, periodically renewed nonrepairable systems.

   (e) Model input

   Structural input parameters: initial number of active and spare modules, number of degradations allowed in the active set, number of good modules in the first safe shutdown state.

   Physical input parameters: failure rate for each active module, failure rate for each spare module, failure rate for each good module in the safe shutdown state, transient fault arrival rate for each active module, and the mean duration of a transient fault.

   Logistical input parameters: (a) number of repair facilities, repair rate for each module when the system is operating in less than full configuration state or is in a safe shutdown state, and restart rate to attain the full-configuration state from the crash-fail state. (b) Renewal process: service interval length, replacement or repair rate per module, restart rate from the crash-fail state, mean system checkout duration, and the total time interval between the initiation of the i-th renewal phase and the resumption of system operation with full

configuration. The total time interval is a random variable estimated using the other parameters.

Detection and recovery for permanent faults input parameters: probability of recovery (coverage) from a spare module failure, coverage for active modules, coverage for active set when spares remain, and coverage for successful final degradation to safe shutdown state.

Detection and recovery for transient faults: the number of recovery phases, recoverability (the conditional probability that the fault is non catastrophic, given that a fault occurs), the failure rate of all the hardware engaged in the execution of the transient recovery processes, the recovery duration vector, and the recovery effectiveness vector.

(f) Model output

Mission-time measures for the system: reliability, mean time to first failure, failure rate, the balance of the unreliability contribution of each subsystem, reliability improvement factor resulting from the redundancy, mission time improvement factor between two competing designs.

Aggregate state probabilities: probability that the system is operating without apparent degradation either in performance or in fault tolerance, probability that the system is operating in any of the degraded states, probability that the system is operating in one of the unsafe states or in states with some failed spares that are unrecoverable, probability that the system is in a safe shutdown state, probability of a catastrophic failure.

Steady-state probabilities for expressing the limiting behavior of repairable and renewable systems; Instantaneous availability; Safety-the probability that the system is in an operational state or in a safe shutdown state; Availability of specified potential level of performance (capacity); Average expected potential performance level (capacity); Frequency of failures; Mean lost computation time due to crashes; Mean lost computation time due to safe shutdowns; Mean down time; Average number of module failures.

(g) Interfaces

- Input: C implementation
- Output: C implementation

7. Use cases

Teaching aid.

8. Assumptions and restrictions

Constant transition rates (failure rates).

Assumption of distinct eigenvalues for the matrix of transition rates that results in a solution of $O(n^5)$ as compared to the classical solution that is $O(n^4)$ (applies only to models of repairable systems).

For nonrepairable (closed) systems with repeated eigenvalues the solution can be found only if it can be guaranteed that the rate matrix is diagonalizable.

B.1.3 BQR CARE

1. Source/Reference: N/A.

   Web: `www.bqr.com`

2. Project status

   The tool is commercial product of the BQR Reliability Engineering. CARE is a part of a larger software product that should enhance the reliability and reduce costs in development of electro-mechanical devices. The company exists since 1989. The project is active and consists of several modules that are described together.

3. License type

   Commercial license conditions are not listed on the website. The CARE tool is split in modules, it is possible to obtain selected parts only. It is possible to obtain a cheaper academic license although its price is not disclosed. However, the academic version has limitations in number of elements/components that can be used.

4. General purpose

   The tool is developed for reliability analysis of electronic, computational and mechanical systems (mechatronics). Although it is applicable to software reliability (some mainstream modeling techniques are present in the tool) the main strength and focus is to provide extended support for reliability assessment in mechatronics. For instance, the tool has modules that model and estimate the changes in reliability caused by aging, power, voltage, current and temperature variations. Also, the tool is shipped together with MTBF libraries for electronic components based on widely used reliability standards: HRD5, MIL-HDBK-217, and Bellcore.

5. Platform: Windows.

6. Model

   (a) Model class: analytical and simulation, IT-level (hardware, mechatronics)

   (b) Model type

   Failure Mode, Effects and Criticality Analysis (FMEA/CA), Fault Tree Analysis, Reliability Block Diagrams.

   For electronic components, different reliability standards/models are used: MIL-HDBK-217F Notice 2, 217Plus, British-Telecom HRD5, Siemens SN-29500, IEC62380 - RDF2000 / UTEC80810, NSWC98 mechanical, Bellcore Issue 6, CNET 2000, GJB299. Component

stress (de-rating) analysis is done based on NAVSEA TE000-AB-GTP-010 Revision 2. Testability (analysis of the coverage level of the built in tests of a system, checking the isolation level of those tests) is done by BIT and ATE.

(c) Model description

The tool handles hardware/software failure trees with no limit on function levels, number of assemblies, or number of failure modes. It is possible to convert the FMECA tree into a fault tree (assuming that the Fault Tree Module is purchased by the user).

RBD basic models are: serial, parallel, K-out-of-N and stand-by with or without repair. Advanced models are network and Markov. The Markov extension allows to simulate complex state of a system, including the transition drivers, repair times, and different states of the sub-blocks. The Network extension allows users to simulate complex networks. Multipoint entries and exits to and from the system are possible. Within a network configuration a sub-block (or connection) can be composed from any of the basic and Markov models. Basic blocks in RBD can choose between following failure distributions: exponential, log-normal, Weibull, uniform, Pareto, Rayleigh and bath-tube.

Repair trees allow calculation of MTTR for each composite block (subtree or node). MTTR is calculated as average (by failure rates) of sub blocks maintenance time (replacement or/and repair time depending on sub block repair type). It is also possible to calculate the maximal confident repair time of a block, supposing lognormal repair time density.

(d) Systems modeled: Repairable and unrepairable systems

(e) Model input: Standard possibilities for FMEA/CA, fault tree and reliability block diagram structures.

(f) Model output

- FMEA/CA: failure mode probabilities
- fault trees: minimal cut sets (with probabilities), probabilities and rates for all events in the tree
- reliability block diagrams: reliability, availability, down-time, MTBCF, MTTR and failure rates, hazard rate, total downtime during mission
- repair trees (based on the FTA methodology): MTTR for each composite node in the tree

(g) Interfaces

- Input: GUI, direct import from CAD/CAE interfaces for MTBF analysis (Mentor Graphics, Cadence, Or-Cad, Excel CSV, ERP SAP, ERP MFG-Pro), data exchange with other BQR products.
- Output: GUI, some modules can exchange data with other BQR products.

7. Use cases

   The company supports projects from the following industry sectors: aviation, automotive, chemical, electronics, consumer electronics, defense, military, energy plants, gas and oil, medical, public transportation and semiconductors. Their major customers are: IBM, Siemens, Lockheed Martin, BAE, Fokker, Rafael, Elisra, Israel Aircraft Industries, Israel Electric Company, Tadiran, Elta. However, since the BQR offers both tools and consulting services, it is unclear which of these companies use BQR tools autonomously.

8. Assumptions and restrictions: N/A.

B.1.4 CARE III

1. Source/Reference: [16], [85], [15], [37]

2. Project status

   The tool was originally developed by NASA together with the Raytheon Company as the successor of CARE, to improve its inadequacies. The last known activity is from the 1984. The project continued to be developed under the name HARP (also described in this work). We survey the tool because of its historical significance.

3. License type: N/A

4. General purpose

   Computer-Aided Reliability Estimation Program (CARE III) is the general purpose modeling tool for reliability assessment of complex, highly available systems.

5. Platform: VAX-11.

6. Model

   (a) Model class: analytical, IT-level.

   (b) Model type: supported models include fault trees, non-homogeneous Markov chains and semi-Markov chains.

   (c) Model description

   Failure behavior is specified using a fault tree and modeled as non-homogeneous Markov chain. Failure treatment and repair are modeled using semi-Markov chains. Simple and double failures are supported.

   (d) Systems modeled: highly reliable non-repairable systems.

   (e) Model input

   - Stage name (identifier for a subsystem made up of modules with the same Weibull time-to-failure distribution)

- Number of initially functioning modules in stage Minimum number of modules for stage operation
- Fault-handling models
- Fault type (either permanent, transient, or intermittent)
- Transition rate from active to benign fault state (benign state: fault manifestation has vanished)
- Transition from benign to active fault state
- Rate at which fault is detected by self-test
- Rate at which a fault generates errors
- Rate at which errors are detected
- Single point failure distributions and parameters for previously defined fault tree description
- User describes relationships between system stages using a fault tree language
- Description language supports AND, OR, M out of N, invert input gates, and up to 2000 total events and 70 date input events
- Hardware and functional redundancy are described

(f) Model output

- Probability that a given module in a stage X has not experienced a specified category fault by time t
- Reliability of a module
- Rate of occurrence of a specified category fault in a given operational module
- Rate of occurrence of faults in the remaining fault-free modules at time t given a specified number of faulty modules summed over all stages
- Probability that a given module has a specified category latent fault at time t given that it has experienced some fault by time t
- Probability that a given module has a latent fault at time t given that it has experienced some fault by time t
- For a given stage, the probability that a subsystem contains a specified number of latent faults given a specified number of faulty modules
- Probability that a system having a specified accumulated number of faults has a specified accumulated number of latent faults
- Probability that a system containing a specified number of faults would be in a supercritical state if a given category fault occurred at time t
- Probability that a system containing a specified number of faults would enter a specified critical state if a given fault occurred at time t
- Probability, given that a system enters a specified critical state at time t, that this event eventually causes a system failure

- Probability that a system having a specified number of faults is in a critical state at time t
- Rate at which systems having a specified number of faults fail at time t due to critical fault conditions
- Probability that the system recovers from a fault

(g) Interfaces

- Input: Interactive interface for model input.
- Output: Textual/tabular results.

7. Use cases: NASA, modeling of avionics systems and computer systems operating in space.

8. Assumptions and restrictions

   The tool is restricted to non-repairable systems. The complexity of models is limited with difficult solving of semi-Markov models. No support for "cold spares" is provided.

B.1.5 CARMS

1. Source/Reference: [166]

   Web: `http://www.tc.umn.edu/~puk/carms.htm`

2. Project status

   The last project update was made in 2001, the tool is still available for download.

3. License type: free download (no license).

4. General purpose

   Computer-Aided Rate Modeling and Simulation (CARMS) is an integrated Markov modeling and simulation tool designed to solve a wide range of time-dependent, prediction-oriented problems. The features include a state diagram-based CAD environment for model setup, a spreadsheet interface for data entry, an expert system link for automatic model construction, and an interactive graphics interface for displaying simulation results. Primary applications are reliability analysis, operations research, fault-tolerant systems, engineering design, and scientific or statistical modeling using Markov analysis.

5. Platform: Windows.

6. Model

   (a) Model class: analytical, IT-level.

   (b) Model type: Markov-chains.

(c) Model description

CARMS uses standard Markov chains. State diagram can be entered graphically. Solution options are then chosen, and results are plotted/displayed. Different numerical methods for Markov chain solving are available. Symbolic tool for expression simplification is also available.

(d) Systems modeled: time-dependant, prediction-oriented system.

(e) Model input

Markov chain specification: states and transition probabilities.

(f) Model output

- Reliability prediction
- Markov analysis
- Maintainability and availability predictions
- Inventory system analysis
- Probability evaluation

(g) Interfaces

- Input: graphical editor for state modeling, tables for transition probabilities.
- Output: plotting, graphical result representation.

7. Use cases: availability analysis, operations research, fault-tolerant system design, state diagram construction, inventory system analysis. No concrete tool users are known.

8. Assumptions and restrictions: N/A.

B.1.6 CASRE/SMERFS

1. Source/Reference: [133]

   Web: `http://www.openchannelfoundation.org/projects/CASRE_3.0`

   `http://www.slingcode.com/smerfs/`

2. Project status

   The last known version if from 2002, the current project status is unknown. There project was, however, further developed under the name Web-CASRE. The modeling part comes from the public domain SMERFS (Statistical Modeling and Estimation of Reliability Functions for Software) package.

3. License type: Open Channel Software License.

4. General purpose

   CASRE (Computer-Aided Software Reliability Estimation) is a tool for automatic measurement and assessment of software reliability. It contains graphical representation of failure data, their processing (e.g., filtering) as well as failure predictions for observed software module.

5. Platform: Windows, Web-CASRE is Web-based application with a Unix back-end (client).

6. Model

   (a) Model class: quantitative/analytical-simulation, IT-level.

   (b) Model type: Jelinski-Moranda model, non-homogeneous Poisson process model.

   (c) Model description

   Jelinski-Moranda model is used to predict time between two failures (MTBF). It is a history-based method using the statistical sampling of previous failures. Non-homogeneous Poisson processes are used to model the number of failures.

   (d) Systems modeled: reparable (software) systems.

   (e) Model input: failure data set (e.g., time between successive failures).

   (f) Model output: number of failures, duration of test interval, failure density, MTBF, cumulative number of failures, reliability.

   (g) Interfaces

   - Input: text (tabular/file) or graphical representation.
   - Output: file or high-resolution plot.

7. Use cases: software testing (unit-testing and acceptance testing).

8. Assumptions and restrictions

   Usability is limited as the input failure data files can be up to 64KBytes (accounts to 3000 data-sets/file or 2000 test intervals). The number of models which can be started simultaneously depends only on the memory size (CASRE simulates all data in the main memory).

B.1.7 CPNTOOLS

1. Source/Reference: [216]

   Web: http://wiki.daimi.au.dk/cpntools/cpntools.wiki

2. Project status

   CPN Tools is continuously developed and maintained by the University of Arhus (Denmark). The latest version is from 2006.

3. License type: commercial, military/government, non-commercial/academic.

4. General purpose

   CPN Tools is used for the graphical development, simulation and analysis of colored Petri-nets. Hence, various distributed processes can be modeled which need to communicate and synchronize with each other.

5. Platform: Windows, Linux – Fedora Core 2 (must support OpenGL hardware acceleration to enable graphical editing).

6. Model

   (a) Model class: analytical/simulation, IT-level.

   (b) Model type: Colored Petri nets.

   (c) Model description

   Colored Petri Net is a graphical oriented language for design, specification, simulation and verification of systems, and represents an extension of the Petri Net model. It is in particular well-suited for systems that consist of a number of processes which communicate and synchronize. Typical examples of application areas are communication protocols, distributed systems, automated production systems, workflow analysis and VLSI chips.

   (d) Systems modeled: distributed synchronized systems.

   (e) Model input

   Colored Petri net to be analyzed.

   (f) Model output

   CP-nets can be analysed in four different ways. The first analysis method is interactive simulation. It is very similar to debugging and prototyping. This means that a CP-net model can be executed, to make a detailed investigation of the behavior of the modeled system. It is possible to set breakpoints and to visualize the simulation results by means of different kinds of graphics, e.g. Message Sequence Charts (also know as event traces).

   The second analysis method is automatic simulation which is similar to the program execution. It allows a fast execution of thousands or millions of transitions. The purpose is to investigate the functional correctness of the system or to investigate the performance of the system, e.g., to identify bottlenecks, to predict the use of buffer space or the mean/maximal service time.

   The third analysis method are occurrence graphs (also called state space or reachability graphs). The basic idea behind occurrence graphs is to construct a directed graph which has a node for each reachable system state and an arc for each possible state change. Obviously, such a graph may become very large, even for small CP-nets. However, it can be constructed and analysed automatically, and there exist techniques which make it possible to work with condensed occurrence graphs without losing analytic power. These techniques build upon equivalence classes.

   The fourth analysis method is place invariants. This method is very similar to the use of invariants in ordinary program verification. The user constructs a set of equations which is proved to be satisfied for all reachable system states. The equations are used to prove properties of the modeled system, e.g., absence of deadlock.

   (g) Interfaces

   - Input: rich GUI for Petri net design.

- Output: rich GUI for analysis/simulation/monitoring representation, additional file export.

7. Use cases: verification and assessment of availability and performance for network protocols, software components, workflows and business processes, flight control systems.

8. Assumptions and restrictions: N/A.

B.1.8 DyQNtool+

1. Source/Reference: [100], [99]

2. Project status

   The tool was developed at the University Twente during the 1990s. The current project status is not known, however, there are indications that the project is probably not under active development.

3. License type: academic (free) license.

4. General purpose

   DyQNtool+ applies the concept of dynamic queuing networks. It enables formal specification of reliability and performability properties, as well as their mutual dependencies.

5. Platform: Unix.

6. Model

   (a) Model class: quantitative, analytical/simulation, IT-level.

   (b) Model type: Markov reward models.

   (c) Model description

   DyQNtool+ is based on the Markov reward models, specified using dynamic queuing networks. The tools uses SHARPE and SPNP modules (see description of respective tools for more information). SPNP is used for model generation and SHARPE for assessment. This combination allows for assessment of stationary, transient and cumulative performability and reliability indicators. DyQNtool+ generates the required Markov chains and SHARPE performance model. The model is solved in SHARPE and combined with state probabilities, derived from SPNP.

   (d) Systems modeled: N/A.

   (e) Model input

   The possible inputs to the tool are:

   - CSPL description of the reliability model (directly importable to SPNP)
   - Description of the parameterized queuing network in form of the C implementation

- Mapping of the SPNP marks to the parameters of the queuing network (C implementation)
- Performance as reward in form of C functions

(f) Model output: reliability and performability in tabular form.

(g) Interfaces

- Input: C implementation.
- Output: C implementation.

7. Use cases: N/A.

8. Assumptions and restrictions: limited to queuing networks.

B.1.9 Eclipse TPTP (Test and Performance Tool Platform)

1. Source/Reference: N/A

   Web: http://www.eclipse.org/TPTP

2. Project status

   The project is active. It belongs to the top-level projects of the Eclipse Foundation. It represents the further development and extension of the Eclipse Hyades project.

3. License type: Eclipse Open-source license.

4. General purpose

   TPTP offers a standardized, generic and extensible tool platform for development of software solutions for performance and reliability assessment, as well as testing tasks. The TPTP project is divided into following domains:

   - Platform: offers a common infrastructure for the user interface, data model, data collection and communication. The given functionalities can be expanded using Extension Points.
   - Testing Tools: offers the basic extensions for performing software testing tasks and development of test environments.
   - Tracing and Profiling Tools: extends the Platform with support for data collection of Java programs that use the Common Trace Model. Additionally, graphical representation and data analysis are offered.
   - Monitoring Tools: offers analysis of the log files using statistical models. Numerous formats are supported, which can be used for performability or reliability analysis.

5. Platform: all platforms with JVM.

6. Model

   (a) Model class: benchmarking, monitoring and test; IT-level.

(b) Model type: TPTP Monitoring Tools, analysis and (pre)processing of logfiles.

(c) Model description

Events are analyzed from logfiles using pattern matching rules and XPath log analyzer.

(d) Systems modeled: Java-based software systems.

(e) Model input: logfiles or database query results.

(f) Model output: vector containing Java objects with solutions (e.g., reliability).

(g) Interfaces

Using the concept of Eclipse plugins, numerous interfaces in form of Extension Points are available for input/output.

7. Use cases: testing, performability and reliability analysis of communication based Java applications (client-server applications).

8. Assumptions and restrictions

TPTP is limited to analysis of Java applications. The extensions for other languages/platform (e.g., C++) have been announced.

B.1.10 ExhaustiF

1. Source/Reference: [65]

Web: http://www.exhaustif.es/

2. Project status

The project is active. The tool is being developed at the Madrid University of Technology, in cooperation with two software SMEs. The tool is available in additional versions as IBM Rational Rose and Eclipse plugins.

3. License type: commercial license

4. General purpose

Exhaustif is a tool designed for black-box and gray-box testing using the SWIFI method (Software Implemented Fault Injection). Design and programming errors can be identified using the tool. The main purpose is to improve reliability and availability properties of software intensive systems during the development process (including integration and system tests).

5. Platform: RTEMS/ERC32 and RTEMS/Intel (Real-Time Operating System for Multiprocessor Systems). It has been announced that versions for Windows/Intel and Linux/Intel are planned.

6. Model

(a) Model class: quantitative - benchmarking and test, IT-level.

(b) Model type: SWIFI (Software Implemented Fault Injection).

(c) Model description

The tool injects faults at the program level (procedures, variables) and at the hardware level (CPU, memory, I/O). The test object is executed automatically in the host-target environment.

(d) Systems modeled: embedded software in distributed heterogeneous systems.

(e) Model input: program source code and parameters for fault injections at the hardware level.

(f) Model output: performability and reliability of the test object, based on the post injection data analysis.

(g) Interfaces

- Input: Java GUI to define the faults injection campaign.
- Output: SQL database to save the test results obtained from system under test.

7. Use cases: Fault Injector Kernel (FIK) for an EADS-Astrium SPARC ERC32-based MCM processor.

8. Assumptions and restrictions: currently no support for general purpose hardware/software platforms.

B.1.11 FAIL-FCI

1. Source/Reference: [199], [104]

2. Project status

The project is active, the last known activity is from the 2006. The tool is developed by INRIA.

3. License type: N/A.

4. General purpose

FAIL-FCI (Fault Injection Language - FAIL Cluster Implementation) is the tool for reliability assessment in the cluster and gird environments. It enables modeling and implementation of failure scenarios using an abstract failure description language and a fault injector. The tool can also be used to test the effectiveness of the existing fault-tolerance mechanisms in the cluster systems. FAIL-FCI also supports stress tests which can be used to quantify non-functional aspects such as robustness and performability.

5. Platform: Linux.

6. Model

(a) Model class: quantitative, benchmarking and test, IT-level.

(b) Model type: software-based fault injection.

(c) Model description

FAIL is the language which enables description of failure scenarios. The FAIL programs are translated by FCI compiler into the C++ source code and linked with FCI libraries. Thus archived source code is then deployed on the grid nodes. On the host node, the program is finally compiled and executed as FCI-Daemon. The distributed application (system under test) is then executed by the FCI-Daemon which instruments the failure scenario.

(d) Systems modeled: distributes systems, grids, clusters.

(e) Model input

Formatted failure scenario description using the FAIL language (in the text form).

(f) Model output

C++ source code with annotations pointing to the faulty behavior of the test object (software-based fault injection, such as probabilistic process termination).

(g) Interfaces

- Input: N/A.
- Output: N/A.

7. Use cases: fault injection into distributed systems, stress testing.

8. Assumptions and restrictions: N/A.

B.1.12 FIGARO / KB3 Workbench

1. Source/Reference: [29], [84], [33], [32], [31], [26], [28], [45], [34], [30], [27], [36]

   Web: http://www.edf.fr/72493m/txt/Home-fr/Research–Development/The-scientific-community/Downloads/KB3.html

2. Project status

   FIGARO/KB3 is developed and used by EDF (Electricite de France). It consists of a model description language (FIGARO), user interface and model solvers (KB3 Workbench) that are external and can be exchanged. All parts are separately available. The project is under active development.

3. License type

   Evaluation license with no time limit, the only limitation is the size of studies that can be created, loaded, etc.: they may not contain more than 80 objects.

   Commercial license is distributed via Apsys (http://www.apsys.eads.net).

4. General purpose

   Evaluation of system reliability, availability, performance using various number of models. The peculiarity of the solution is that knowledge bases are used to enable abstract system modeling. Different compilers and translators then automatically infer the data which are necessary for the classical reliability model (e.g., for a Markov chain or a Petri net).

5. Platform: Unix/Xwindow, Windows.

6. Model

   (a) Model class: analytical and simulation, IT level.

   (b) Model type: Markov chains, Boolean logic Driven Markov Process (BDMP), fault trees, reliability block diagrams, Petri nets.

   (c) Model description

   A system is modeled using the abstract FIGARO modeling language which supports object-oriented model. Using predefined knowledge base, the system then transfers this description into the appropriate availability model, such as Petri net or reliability block diagram. The model can be further solved using the following techniques: Markov chain generation and quantification (analytic), Monte Carlo simulation or sequence generation and quantification. Different model analysis tools can be plugged in to perform these tasks.

   (d) Systems modeled: repairable and non-repairable systems.

   (e) Model input

   For each class, its attributes are defined, together with the failure model and its parameters, occurrence of failure and repair and interactions. After such a model is built, classes are instantiated as objects that represent the system. Graphical user interface is provided for this purpose. System then automatically transfers an input model into the working model, using the appropriate knowledge base.

   KB3 Also features AR2FIG0 converter, which can accept definitions in AltaRica Dataflow language and convert them to FIGARO. Thanks to AR2FIG0, it is possible to take advantage of the computing power of the tools FIGSEQ, YAMS and FIGMAT-SF, and use the already existing models in KB3.

   (f) Model output

   - Reliability and transient reliability (Markov chains)
   - Reliability, availability, average performance, number of events, time spent in state classes (simulations)
   - Reliability of repairable and non-repairable systems (sequence generation and quantification)

   (g) Interfaces

- Input
  FIGARO offers object-oriented language for defining model input. Graphical user interface is also provided for creating model descriptions which generates ASCII files as output (they are used as inputs for analysis tools). User interface can also perform model checking.
- Output
  KB3 offers different outputs: raw data in text files or automatic shaping (graphs, images).

7. Use cases

   Known use cases include availability studies performed to optimize the design or the exploitation of industrial systems: production installations, public networks (electricity, transport, telephony), reliability assessment of high technology systems (on-board systems, medical devices...), and the control of the safety of risky industrial processes, e.g. in nuclear, chemical, petrochemical industries.

8. Assumptions and restrictions

   The number of states grows exponentially when the size of the system increases (Markov chains). The generation of confidence intervals on the probabilities of rare events that are not Markov modeled is time consuming. Finally, generation of sequences by inferences can yield only reliability.

B.1.13 GRAMP/GRAMS

1. Source/Reference: [69], [80], [81]

2. Project status

   GRAMS development started as a PhD thesis [80] and was further extended based on ARIES, SURF and CARE III tools. The tool is not being maintained anymore.

3. License type: the last known implementation is from 1984. Current status is unknown. License type is unknown. We survey it for historical reasons.

4. General purpose

   GRAMP tool is used for modeling the following aspects of a fault-tolerant system: coverage, preventive maintenance, acquisition cost, operations cost, support cost, and provides sensitivity analysis. GRAMS tools is used for reliability, maintainability, and life-cycle cost prediction of the system modeled by GRAMP.

5. Platform: FORTRAN 77 implementation for VMS.

6. Model

   (a) Model class: simulation, IT level.

(b) Model type: Continuous-time Markov Model (GRAMP) solved by Monte Carlo discrete event simulator (GRAMS).

(c) Model description

GRAMP and GRAMS offer methodology for verifying reliability, maintainability and cost specifications for fault-tolerant systems during the concept and design stages. This method systematically and quantitatively factors the design, maintainability and support into the analysis while accounting for uncertainties through front-end sensitivity analyses. System is modeled using continuous-time Markov model, and solved using simulation. Time varying failure rates are handled as piecewise constant failure rates.

(d) Systems modeled: non-repairable and repairable systems.

(e) Model input

GRAMP supports modeling of systems, subsystems and modules. At the system level, the inputs are: Fixed charge for repair; Failure charge for breakdown; Run options; Print options; Mission length; Acquisition cost (system level and module level); Average repair cost. At the subsystem level the inputs are: Number of modules in subsystem; Reliability requirement; Opportunistic maintenance option. (When one module is repaired, this parameter allows the user to specify whether the maintenance strategy involves searching for other failed modules and repairing those too.); Subsystem structure (critical sets). At the module level, the inputs are: Preventive maintenance level; Redundancy level; Replacement level; Sensitivities to compute; Failure rates; Coverage-active and spare units.

GRAMS supports the following input parameters at the system level: Costs associated with various maintenance actions; Cost of required maintenance on a backup system when it fails its permission inspection; Cost incurred when the switching mechanism for the backup system fails after the primary system has failed; Cost of opportunistic maintenance done on one or more components that were repaired while the host was in shop for reaching its maximum operating times (MOT) or for on-schedule maintenance; Cost incurred whenever scheduled line replaceable units (LRU) maintenance is done; Cost incurred whenever unscheduled LRU maintenance is done; Costs associated with host removal, shipment to shop, and repair for scheduled maintenance and for unscheduled maintenance. For the system level input the following event probabilities have to be given: Probability of mission type A; Probability of doing end of mission even when not required; Probability of not doing end of mission maintenance even when required; Probability of discrete events (such as maximum temperature and lightning events); Probability of each level of severity (n of them) for the discrete events (such as maximum temperature and lightning events); Probability of host failure on a mission; Probability of backup inspection failures; Probability

of backup switching failures. Furthermore, Number of host operating hours per life and per year; Lengths of mission A and B in hours; Number of severity levels (n) for discrete events; Number of hosts to be simulated; Number of hosts in the fleet; Array of times for failure rate data; Host population and the minimum and maximum number of hosts, can also be specified at this level.

(f) Model output

GRAMP supports the following output options: Cost evaluator model (CEM): Failure/maintenance events per million hours, Mean time between events, Relative operations and support (0 & S) cost, Acquisition cost, Weight, Sensitivities. Reliability feasibility model (module or subsystem basis): Reliability, Mean times to failure, Reliability time history plot. System bookkeeping: Reliability, MTTF, 0 & S and acquisition costs, Total cost of ownership, Weight, CEM events per million hours (abort rate), Mean time between failures (MTBF) and mean time between repair (MTBR).

GRAMS supports the following outputs options: Namelist; Parameter definitions; Subsystem definitions; Module definitions; System results; Subsystem reliability; Subsystem MTBF; Module MTBF due to component failure/module MTBF due to coverage failure; System MTBR by maintenance type; MTBR driven by subsystem maintenance (average over host life); MTBR driven by subsystem maintenance (yearly); Events driven by subsystem maintenance (yearly average over host life); Events driven by subsystem maintenance (yearly); 0 & S cost breakdown; Component replacements by maintenance type (yearly average over life cycle); Component replacements by maintenance type (yearly); Component replacements attributed to maintenance plan (yearly and aggregate); Backup failures; Print option definitions (determines which outputs are selected for printing).

(g) Interfaces

- Input: FORTRAN 77 implementation
- Output: FORTRAN 77 implementation

7. Use cases

GRAMP and GRAMS have been applied to several phases of design projects for fault-tolerant electronic controllers. They were used to determine if the improvements in reliability, safety and function (such as decrease in fuel consumption) were worth the overall cost of producing an electronic controller. In addition, a sensitivity analysis was performed to determine the effects of uncertainties in coverage, dissimilar redundancy, hardware redundancy and maintenance on reliability and cost. During the final design selection phase, CRAMP and GRAMS were implemented as analysis tools tor designing controller configurations to meet required reliability and safety specifications. Reliability and maintainability goals were also evaluated based on engineering design constraints.

The objective of the FAFTEEC program was to develop a design approach for full-authority digital electronic control systems with reliability the primary consideration factor. The approach used in attacking this objective was to identify a baseline full-authority digital electronic control system for and advanced fighter aircraft and then improve on this baseline control with respect to specific goals using redundancy, recovery strategies, and maintenance philosophies. Candidate control designs were evaluated with respect to cost and weight in addition to their ability to satisfy the design goals. The baseline control system was modularized to yield identifiable components (pumps, thermocouples, actuators, etc.). For these components, reliability and cost information was accumulated. Many of these configurations were screened with GRAMP. GRAMS tested promising configurations from GRAMP, using a time-varying analysis approach based on Monte Carlo techniques. The results of the GRAMP and GRAMS analysis showed necessary cost and weight increases associated with achieving an order of magnitude improvement in mission reliability by using a fault-tolerant structure as opposed to the baseline system.

8. Assumptions and restrictions

   The tool can handle a maximum of 1500 Markov states. Emphasis appears to be on military systems rather than commercial systems. The system has following additional assumptions: Randomly deteriorating (stochastic failure process); Independent component failures; (Piecewise) constant failure rates; State transition immediately following component failure; Components are either working or failed. Furthermore, the following maintenance assumptions exist: Stationary maintenance policies; Repair brings component back to new condition; Maintenance actions at instant of component failure; Instantaneous repair; unlimited service capacity; When a subsystem fails, fix everything; Maintenance policy restrictions from Coherent System Repair Models. Finally, the time clock assumption supports only continuous-time, single failure per state transition.

B.1.14 HARP

1. Source/Reference: [70], [86], [17], [18]

2. Project status

   HARP (Hybrid Automated Reliability Predictor) has been continuously developed at the Duke University from 1986-1994 until version 7.0. Since then there have been no new versions.

3. License type: N/A.

4. General purpose

   HARP is a tool for reliability and availability assessment using Markov chains. It uses fault trees (or user-specific graphical representation) as the input mechanism.

5. Platform: MS/PC-DOS, Microsoft Windows NT, OS/2, DEC VMS and Ultrix, Berkeley UNIX 4.3 and AT&T UNIX 6.2.

6. Model

   (a) Model class: analytical and simulation, IT-level.

   (b) Model type

   The tool supports three model types. FORM (Fault Occurrence and Repair Model) contains the information about the hardware structure, fault-occurrence processes and manual (off-line) repair. These can be specified either as Markov chains or fault trees. FEHM (Fault/Error Handling Model) describes permanent, sporadic and transient faults, as well as description of automatic (on-line) repairs. Markov chains are used to specify these.

   (c) Model description

   HARP uses a hybrid model (FORM and FEHM) to express statements about reliability and availability. Based on these information, Markov chain is automatically generated and solved.

   (d) Systems modeled: repairable and non-repairable systems.

   (e) Model input

   The following input parameters are supported: fault tree representation in a graphic form, Markov model representation in a graphic form, failure rates with variation band, repair rates, initial conditions, near-coincident faults by location, near-coincident faults by component, and description of the recovery processes form in terms of a Petri net.

   (f) Model output

   The tools calculates reliability (unreliability) and instantaneous availability (unavailability) with respect to time. Additionally, sensitivity of the reliability or availability prediction to parameter variations and initial state uncertainty can be assessed. This information provides a measure of the system's sensitivity to design faults. Finally, failure probabilities attributed to exhaustion of redundant element, single-point of failure, and near-coincident faults can be determined.

   (g) Interfaces

   - Input: textual (file) or graphical user interface. There are interfaces for CARE III and ARIES, enabling HARP to read their models, too.
   - Output: tabular in textual (file) form.

7. Use cases: N/A.

8. Assumptions and restrictions

   HARP does not enable modeling of repair strategies. Furthermore it is not possible to generate models with time dependence or "cold spares".

B.1.15 IsoGraph FaultTree+

1. Source/Reference: [108]

   Web: http://www.isograph-software.com/ftpover.htm

2. Project status

   Commercial tool in active development. Latest full version is V11 from 2005, since then version 11.1 (2007) is also available.

3. License type: commercial license, price is not openly disclosed. Evaluation version is available.

4. General purpose

   The tool offers reliability analysis that allows fault and event tree analysis to be performed. Customized Markov models may be linked to an event in the fault or event tree diagram. Independent analysis of fault trees, event trees and Markov models is also possible.

5. Platform: Windows.

6. Model

   (a) Model class: analytical and simulation, IT-level.

   (b) Model type: fault trees, event trees, Markov models.

   (c) Model description

   The fault tree method involves the creation of a fault tree diagram composed of gates and basic events that represents the logical description of a system failure, known as the TOP event, in terms of the failure of the components that comprise the system. After creating the diagram the user assigns failure characteristics of the system components. On completion of the model the system analysis is performed. To do this the FaultTree+ software first determines the minimum combinations of component failures that will cause a system failure, these are known as the minimal cut sets. Finally FaultTree+ calculates the quantitative parameters such as system unavailability and failure frequency. FaultTree+ includes an event tree analysis option. The event tree model may be created independently of the fault tree model or may use fault tree analysis gate results as the source of event tree probabilities. FaultTree+ also allows the user to construct Markov models for use as the source of basic event data. The Markov models may be analysed independently of the fault tree analysis.

   (d) Systems modeled: repairable and non-repairable systems.

   (e) Model input

   The fault tree model supports the following input parameters:

   - gates and events

- gate parameters: type (OR, AND, VOTE, NOT, XOR, INHIBIT, PRIORITY, TRANSFER or NULL), name and description, retain results (minimal cut sets and quantitative results retained for the gate during analysis), gate and event inputs, notes, hyperlinks to external documents
- event parameters: failure model (RATE, FIXED, MTTF, DORMANT, SEQUENTIAL, ET INITIATOR, STANDBY, TIME AT RISK, BINOMIAL, POISSON, RATE/MTTR, WEIBULL, FIXED-PHASED, RATE-PHASED), name, description, CCF (common cause failure) model, logic mode, order of failures, generic failure model, notes, hyperlinks

The event tree model may be created independently of the fault tree model or may use fault tree analysis gate results as the source of event tree probabilities.

The input for Markov models are discrete system states and transitions (also time dependent). The models created in the Markov analysis module may be linked to basic events in the fault tree and event tree analysis modules.

(f) Model output

The fault tree module generates the following output: CCF analysis, importance analysis (Fussel-Vesley, Birnbaum, Barlow- Proschan, sequential importance measures), uncertainty (confidence levels can be determined from event failure and repair data uncertainties) and sensitivity analysis (allowing automatic variation of failure and repair data between specified limits), time dependent analysis (providing intermediate values for time dependent systems parameters), fault tree house event analysis, full minimal cut set analysis, upper bound calculations, risk analysis.

The event tree module generates the following output: risk calculation, full minimal cut set analysis, risk importance analysis (identifying major risk factors), sensitivity analysis (automatic variation of event failure and repair data between specified limits).

Finally, the Markov module generates the following output: unavailability, availability, unreliability, reliability, failure frequency (unconditional failure intensity), repair frequency (unconditional repair intensity), conditional failure intensity, conditional repair intensity, number of expected failures, number of expected repairs, mean unavailability over lifetime, mean availability over lifetime, expected total downtime over lifetime, expected total uptime over lifetime.

(g) Interfaces

- Input: all three modeling modes (fault tree, event tree, Markov model) use rich Windows-based GUI that enables model construction, edit and management.
- Output: all Isograph tools use integrated Report Generator as output interface. The Report Generator is separately described.

7. Use cases: aerospace, defense, automotive, nuclear, rail, chemical process plant, oil, gas and medical industries.

8. Assumptions and restrictions: N/A.

B.1.16 IsoGraph AvSim+

1. Source/Reference: [107]

   Web: http://www.isograph-software.com/avsover.htm

2. Project status: last version is V10 from 2006, the tool is under active development.

3. License type: commercial license, price is not openly disclosed. Evaluation version is available.

4. General purpose: availability and reliability simulation of complex and dependent systems.

5. Platform: Windows.

6. Model

   (a) Model class: simulation, IT-level.

   (b) Model type: fault tree, network (reliability block) diagram.

   (c) Model description

   The logical interaction of failures (and how they affect the system performance) is modeled using fault trees or network (reliability block) diagrams. They are used to model failure and success or levels of throughput in the system. Consequences are assigned to any level of the logical diagrams to indicate the effects of failures (financial, operational, safety or environmental). Labor, spares and failure data can be imported with any operational phase information and task group assignments. The tool analyses the system using Monte Carlo simulation to provide availability and reliability parameters, life cycle costs, importance rankings. Spare holdings and maintenance intervals can also be optimized.

   (d) Systems modeled: repairable and non-repairable systems.

   (e) Model input

   The models support following input parameters: network elements, fault tree elements, failure modes, maintenance models, data sets (time to failure), consequences, spare parts, labor categories, equipment, task groups, phases.

   (f) Model output

   The tool supports several output options:

- Historical data analysis: the tool offers Weibull analysis of historical failure and repair data by assigning probability distributions which represent the failure or repair characteristics of a given failure mode. The failure distribution assigned to a given set of times-to-failure (data set) may be assigned to failure models which are attached to blocks in a network diagram or fault tree. The Weibull module then analyses time-to-failure and time-to-repair using various distributions (e.g., exponential, 1-, 2- and 3-parameter Weibull, Bi- und Tri- Weibull, lognormal, normal, Weibayes, phased Bi- und Tri- Weibull). The Weibull module automatically fits the selected distribution to the data provided and displays the results graphically in the form of cumulative probability plots, unconditional probability density plots and conditional probability density plots.
- Spares optimization: AvSim+ can be used to simulate the effects of different spares holding levels on lifetime costs. The program performs simulation runs for each combination of spare part holdings for each selected spare part. Once all the simulation runs have been completed AvSim+ will display the optimum spare holdings from a cost viewpoint at site and depot.
- Maintenance optimization: AvSim+ can be used to determine whether it is worthwhile performing planned maintenance or inspections on components, and if so, what the optimum maintenance interval should be. AvSim+ locates the optimum interval for planned maintenance and inspection tasks by varying the maintenance interval and repeatedly simulating the lifetime costs.
- Simulation watch: the simulation watch facility is designed to allow the user to check the logic of the system availability model. During a 'simulation watch' the program proceeds with the simulation on a 'step by step' basis and lets the user view the status of the system at each step. The simulation process moves forward in time but halts when there is a change of event.
- Lifetime costs and production capacity: cost and production capacity can be modeled, apart from availability and reliability. Labor, spares and other costs are taken into account. Consequences can be assigned to system failures allowing the cost of failures to be included in the calculation.
- Safety, environmental and operational impact: by allocating safety, environmental, and operational consequences to selected system failures, frequency and duration of each type of consequence can be determined. Severity values may also be assigned, so that safety, environmental and operational criticality can be determined over the system lifetime.

(g) Interfaces

- Input: AvSim+ allows construction of fault tree or network dia-

grams (reliability block diagrams) using drag and drop facilities. If fault trees are used, AvSim+ will automatically organize the diagram based on the logical connections. For reliability block diagrams, the blocks are placed on the screen, logical connections are made, and the program will automatically deduce the failure logic of the system. Element properties are defined using wizard/Windows-based GUI.

- Output: all Isograph tools use integrated Report Generator as output interface. The Report Generator is separately described.

7. Use cases: aerospace, defense, automotive, nuclear, rail, chemical process plant, oil & gas and medical industries.

8. Assumptions and restrictions: N/A.

B.1.17 IsoGraph Reliability Workbench

1. Source/Reference: [110]

   Web: http://www.isograph-software.com/rwbover.htm

2. Project status: last version is V10.1 from 2007, the tool is under active development.

3. License type: commercial license, price is not openly disclosed. Evaluation version is available.

4. General purpose

   Reliability Workbench is an integrated environment for performing reliability prediction, maintainability prediction, Failure Mode Effect and Criticality Analysis (FMECA), reliability block diagram (RBD) analysis, reliability allocation, fault tree analysis, event tree analysis and Markov analysis. As reliability block diagrams, fault trees and Markov analysis have already been discussed (Reliability Workbench offers an unified GUI for the tools), we will concentrate here on reliability prediction, maintainability prediction and FMECA.

5. Platform: Windows.

6. Model

   (a) Model class: various prediction models, hierarchical block diagram.

   (b) Model type: analytical and simulation, IT-level.

   (c) Model description

   Failure prediction methods define the systems and conditions in which they operate (e.g., temperature), the prediction algorithm then carries out the failure rate calculation as defined by the standard and gives the result. The tool supports the following failure prediction models (standards): MIL-HDBK-217, Telcordia SR-332,

NSWC-98, IEC TR 62380, GJB/Z 299B. Additional models can be added.

A Failure Mode, Effects and Criticality Analysis (FMECA) is a procedure for identifying potential failure modes in a system and classifying them according to their severity values. FMECA is usually carried out progressively in two parts. The first part identifies failure modes and their effects (Failure Mode and Effects Analysis). The second part ranks failure modes according to the combination of severity and the probability of that failure mode occurring (Criticality Analysis).

(d) Systems modeled: repairable and non-repairable systems.

(e) Model input

Model inputs can be classified into three distinct groups: reliability allocation, failure prediction and FMECA.

During the design phase of a product, it is often required to evaluate the reliability of the system. The question of how to meet a reliability goal for the system arises. Reliability allocation can be used to set reliability goals for individual subsystems so that this goal is met. The simplest method is to distribute the reliability goal equally among all subsystems. However, this rarely gives the best distribution of reliability objectives for all subsystems. It is better to allocate reliability values between the subsystems based on complexity, criticality, achievable reliability, or any other factors that are deemed appropriate. The Allocation module within Reliability Workbench offers six methods for assigning subsystem reliability values: non-restricted equal allocation, non-restricted graded allocation, non-restricted proportional allocation, non-restricted redundancy proportional allocation, non-restricted reliability re-allocation, and restricted direct research allocation. Inside the module, a system hierarchy can be constructed where sub-systems may be broken down further to component level, allowing a complete system allocation model.

For the purpose of failure prediction, the components that make up a system can be defined in a tree structure. The tree is composed entirely of components or is subdivided into blocks each of which holds other blocks or components. In this way the system and its sub-systems are easily represented. The failure rate model for each component is made up of a base failure rate for that particular type of component and multiplying factors known as pi-factors. These factors depend on the operating conditions experienced by the component.

A large proportion of data entered when performing a FMECA is descriptive text. The FMECA Module provides a master phrase library which contains commonly used descriptions of component parts, failure modes and effects. These phrases may be inserted into descriptive fields by selecting the required phrase from the library

saving considerable time and ensuring consistency. Users may build up their own phrase libraries or add to the master library. The following inputs are supported: definition of the system to be analysed, construction a hierarchical block diagram, identification of failure modes at all levels of indenture, effects of the failure modes, severity categories of the effects, other failure mode data such as failure detection methods or failure rates, and failure mode ranking in terms of severity and criticality.

(f) Model output

Failure prediction module offers the following output: failure rate prediction, maintainability prediction, mean time to repair prediction, effects of temperature, stress and environment changes on system, sub-system, or unit failure rate prediction.

FMECA module offers the following output: failure effects, severity values and failure causes through the system hierarchy.

(g) Interfaces

- Input: failure prediction module uses graphical user interface that enables selection of failure prediction model and dynamic input of relevant parameters. FMECA module uses GUI for hierarchical block diagram construction. Possibilities for structuring, property, inheritance and encapsulation modeling are provided.
- Output: all Isograph tools use integrated Report Generator as output interface. The Report Generator is separately described.

7. Use cases: aerospace, defense, automotive, nuclear, rail, chemical process plant, oil & gas and medical industries.

8. Assumptions and restrictions: N/A.

B.1.18 IsoGraph Network Availability Program (NAP)

1. Source/Reference: [109]

   Web: http://www.isograph-software.com/napover.htm

2. Project status: version 1.0 from 2005, since then, no new versions available.

3. License type: commercial license, price is not openly disclosed. Evaluation version is available.

4. General purpose

   The Network Availability Program (NAP) enables the prediction of the availability and reliability of communication networks.

5. Platform: Windows.

6. Model

(a) Model class: analytical and simulation, IT-level.

(b) Model type: extended reliability block diagram.

(c) Model description
The NAP network availability model utilizes an extended reliability block diagram methodology that addresses the specific characteristics of the network elements and their connections. In addition to predicting network availability, NAP also provides criticality rankings that identify weak spots in the network.

(d) Systems modeled: communication networks and their elements.

(e) Model input
NAP recognizes the failure logic of a network from the network diagram entered by the user. The diagram represents how individual network element failures interact with other network element failures to prevent data flow between source and target nodes in the network. A network diagram entered into NAP specifies the possible communication paths between different network elements. The paths between network elements, which are defined with connections, may be directional or nondirectional. A directional connection indicates that data may flow in one direction only. The network diagram contains block symbols normally representing the network elements and the cables connecting the elements together.
Parameters for each network element are type, failure data format, failure rate, default MTTR, path logic.

(f) Model output
The NAP tool can calculate the following network parameters: network unavailability, availability, failure frequency, MTTF, MTTR, MTBF, unreliability, reliability, total down time.

(g) Interfaces

- Input
GUI is used for creating extended reliability block diagram. The NAP library allows the construction of a custom library of parts and alternative parts lists using a hierarchical structure. The parts may be dragged and dropped into a network element or network block diagram. Projects may also be attached to a central part library allowing automatic updates to part data as an option. Alternative part lists may also be added to the library allowing selection of the appropriate part type within a network element.
- Output: all Isograph tools use integrated Report Generator as output interface. The Report Generator is separately described.

7. Use cases: communication networks.

8. Assumptions and restrictions: the network structure cannot be discovered automatically, on-line, but has to be imported (modeled) manually instead.

B.1.19 IsoGraph AttackTree+

1. Source/Reference: [106]

   Web: http://www.isograph-software.com/atpover.htm

2. Project status: version 1.0, undated. The current tool status is unknown.

3. License type: commercial license, price is not openly disclosed. Evaluation version is available.

4. General purpose

   Attack trees allow the modeling of threats against the system security. For example, the effectiveness of internet security, network security, banking system security, installation and personnel security may be modeled using attack trees.

5. Platform: Windows.

6. Model

   (a) Model class: analytical/Simulation, service level.

   (b) Model type: attack tree (a variant of fault trees).

   (c) Model description

   AttackTree+, through the use of attack tree models, allows the modeling of the probability that different attacks will succeed. AttackTree+ also allows definition of the indicators that quantify the cost of an attack, the operational difficulty in mounting the attack and any other relevant quantifiable measure that may be of interest. Different categories and levels of consequence may also be assigned to nodes in an attack tree. A successful attack may have financial, political, operational and safety consequences. A partially successful attack may have a different level of consequence to a totally successful attack.

   (d) Systems modeled: attack-prone, mission critical IT systems and services.

   (e) Model input

   The process involved in constructing the attack tree of the target system begins with the identification of the goals of possible attacks. Due to the fact that different attacks may contain similar methods, this may result in inter-linked attack trees. The next stage is to identify all possible attack methods that may result in the goals being achieved. These first two stages will produce the top level of the system model.

   Each attack may consist of a large number of conditions that need to be met to allow the attack to be successful, or it may simply consist of a single quantifiable event. Therefore, the next step in the construction of the attack tree is to break down each attack into the basic conditions. This will result in a full attack tree structure with every 'branch' ending in a single quantifiable event.

Once the structure is complete, it is necessary to specify the likely frequency for each attack. Next, each event probability must be specified, that is, how probable each aspect of the attack is to succeed.

In addition to analyzing how likely an attack is to succeed, attack trees can also employ indicators to describe the cost, whether any special equipment is needed, etc. A value for each indicator type is assigned to each event. Finally, AttackTree+ allows the definition of consequences which can be attached to any gate within the tree. In this way, it is possible to model the consequences of successful attacks on the target system.

(f) Model output

The tool evaluates the following output parameters: minimal cut sets for each gate, probability of each cut set, indicator values for each cut set, probability of each gate, indicator values for each gate, minimal cut sets for each consequence, probability for each consequence, and risk values for each consequence category.

It is also possible to view all combinations of events that will lead to a successful attack, ranked in the probability of success. This list may be filtered according to indicator values (e.g., cut sets where the cost to an attacker is low). It is possible to 'prune' the attack tree to easily identify the route by which attack would succeed. Also, importance rankings may be viewed to identify how security may be improved most efficiently.

(g) Interfaces

- Input: GUI is available that enables attack trees construction and description. Once the gates and events have been described, indicators, consequences and probabilities can be added.
- Output: all Isograph tools use integrated Report Generator as output interface. The Report Generator is separately described.

7. Use cases: Internet applications, banking, networks.

8. Assumptions and restrictions: N/A.

B.1.20 IsoGraph Report Generator

Isograph Report Generator is used as the output interface for all Isograph tools. It supports the following output forms: text, graphs and diagrams. Text report is specified by selecting parts of the project database that should be included in the report (using SQL statements). Graph reports can contain plots with axes, labels, legends and titles that are configured using simple dialogs. Graphs can be 2D or 3D, and their position can be defined. Using diagram reporting, it is possible to include diagrams such as reliability networks or fault trees in the output report. Reports can be generated in various data formats, such as RTF (for reports using graphs and diagrams) or as comma separated values (CSV)

files (for reports containing only text). The last option enables flexible further parsing and processing.

B.1.21 MARK

1. Source/Reference: [123], [122], [12], [184], [187]
2. Project status

   The first tool version was proposed in 1983, the last known application is described in 1993. The project is not active.
3. License type: N/A.
4. General purpose

   Evaluation of reliability of complex systems whose characteristics can be modeled using Markov chains. The tool calculates state probabilities as functions of time, MTBF and average state occupancy probability.
5. Platform: PL/1.
6. Model
   (a) Model class: analytical, IT-level.
   (b) Model type: discrete-state continuous-time Markov models.
   (c) Model description
   The model covers Markov processes where state space is discrete and the time parameter is continuous, that is, state transition can happen in a continuous time-space.
   (d) Systems modeled: any non-repairable systems whose characteristics can be modeled using a Markov chain.
   (e) Model input
   The model is specified using the following parameters: number of states in the model, description of each state, occupancy probabilities at same initial time for each state and transition rates between the states (defines the interconnection between states). The time span for which Markov chain should be solved can also be defined. Finally, commands to merge the results of the models to obtain the system state probabilities can be given.
   (f) Model output
   The primary model outputs are plots of various state probabilities as a function of time (e.g., plot of probability of being in a given state or the probability of not being in a given state for a specified model), plots of MTBF and plots of average state occupancy probability.
   (g) Interfaces
   - Input: N/A.
   - Output: N/A.

7. Use cases

   The known use cases are reliability prediction for the fault-tolerant multiprocessor (FTMP) developed for NASA, airplane systems and safing systems in nuclear reactors.

8. Assumptions and restrictions

   The following restrictions apply: the tool considers exponential distributions only, the coverage cannot be specified, there is no support for repairable systems, and transient and intermittent faults are not addressed.

B.1.22 METFAC

1. Source/Reference: [54], [46], [47], [48], [49], [50], [51], [52], [53], [55]

   Web: http://dit.upc.es/qine/tools/metfac/

2. Project status

   METFAC has been first developed in 1986 as part of a Ph.D. Dissertation [54]. It has been actively and continuously improved ever since. The latest version is from January 2007.

3. License type

   The METFAC tool is copyrighted software, but is offered free to individuals and academic/basic research institutions for educational use or academic non-profit basic research. Four types of licenses are offered:

   - A free license of a version of the tool with limited functionality (continuous-time Markov models with no more than 500 states can be generated/solved)
   - A free license of a 30-day evaluation fully functional version of the tool
   - A free license of a fully functional version of the tool for individuals or academic and basic research institutions for educational purposes and academic non-profit basic research
   - A regular license for a fee of 5,000 euros of a fully functional version of the tool for individuals and academic/basic research institutions for uses other than educational or academic non-profit basic research or for non-academic/non-basic research institutions

   In all cases, the license is granted for specific operating system/hardware platform combinations and to be used on a single hardware platform as identified by the MAC number, which has to be provided by the licensee.

4. General purpose: analysis of performance, dependability and performability of complex systems through rewarded continuous-time Markov chain models.

5. Platform: Linux kernel 2.4.4 and above; Solaris 2.x; and, on demand, any other UNIX variant with C-shell and an ANSI/ISO (standard 89/90) C compiler.

6. Model

   (a) Model class: analytical, IT-level.

   (b) Model type: finite continuous-time Markov chain models with reward rates associated with their states.

   (c) Model description

   METFAC is a software tool for the analysis of performance, dependability and performability of complex systems through rewarded continuous-time Markov chain models. It allows the specification of arbitrary finite continuous-time Markov chain models with reward rates associated with their states and offers several numerical methods for the computation of seven measures on the resulting stochastic reward process.

   (d) Systems modeled: repairable systems.

   (e) Model input

   METFAC offers a model description language which is based on the derivation rules. The language has the following expressive capabilities and keywords:

   - Structure: action, new_state, parameters, state_variables, response.
   - Properties: production_rules, response, reward_rate, start_rate, with_prob, with_rate, initial_probability.
   - Program flow: end, if, external, no, yes.
   - Types: int, double.

   The language further supports arithmetic expressions as well as relational, binary arithmetical, logical, incremental, decremental and cast operators.

   (f) Model output

   The tool offers the following model outputs:

   - Expected transient reward rate
   - Expected steady-state reward rate
   - Expected averaged reward rate
   - Cumulative reward complementary distribution
   - Interval availability complementary distribution
   - Expected cumulative reward till exit of a subset of states
   - Cumulative reward distribution till exit of a subset of states

   (g) Interfaces

   In the current version, the tool has no graphical user interface. However, it has been announced that the next version will offer a Windows-based GUI.

- Input
  Textual files are used for model input: files that specify the model (.spec) and optional C file (.c). The .spec file describes the model using the language and keywords given above. The .c file consists of definitions for all external model-specific functions which can be used within the .spec file.
- Output
  The output is generated in the two-phase model compilation process. In the first pass the .spec file is translated into the .c file. In the second pass, the .c file is compiled and executed. Interactive execution is also possible.

7. Use cases

   The known use cases include a reliability model of a 5-level RAID storage subsystem, a reliability model of a storage system, a performability model of a multiprocessor system, grid cluster computing systems, storage area networks and communication networks.

8. Assumptions and restrictions: N/A.

B.1.23 METASAN

1. Source/Reference: [175], [150]

2. Project status

   METASAN (Michigan Evaluation Tool for the Analysis of Stochastic Activity Networks) is the predecessor of the UltraSAN and Möbius tools. The tools is not in the active development, as the project was assimilated by the Möbius. We survey it for the historical reasons.

3. License type: N/A.

4. General purpose

   Evaluation of the performance and availability using stochastic activity networks with analytical and simulation methods.

5. Platform: Unix.

6. Model

   (a) Model class: analytical and simulation, IT-level.

   (b) Model type: stochastic activity networks (SAN).

   (c) Model description
   Stochastic activity networks are extensions of Petri nets. For more information, see Section 3.1.6 in the Chapter 3: Availability and Performability Models, as well as descriptions of the UltraSAN and Möbius tools in this chapter.

   (d) Systems modeled: repairable and non-repairable systems.

(e) Model input

A stochastic activity network is specified using the SANSCRIPT description language. It allows for modeling of the initial marking, coverage, distribution, trigger times, failure distribution and halting conditions.

(f) Model output: availability, performance and performability of the system under study.

(g) Interfaces

- Input: the input is specified in two files, one containing the model description in SANSCRIPT, the other containing parameters (data) required for the analytical model evaluation.
- Output: calculated results (availability, performance or performability) are stored in a file.

7. Use cases: N/A.

8. Assumptions and restrictions: computation complexity rises fast with the model size.

B.1.24 Möbius

1. Source/Reference: [62], [174]

   Web: http://www.mobius.uiuc.edu/

2. Project status

   The Möbius project is one of the major research projects of the Performability Engineering Research Group (PERFORM) in the Center for Reliable and High-Performance Computing at the University of Illinois at Urbana-Champaign. Research on Möbius has been supported by Motorola, by the National Science Foundation, and by DARPA grant. The Möbius project is based on previous work on the MetaSAN and UltraSAN tools, which were originally developed to model and solve stochastic activity networks (SANs). The project is still active and its development continues.

3. License type: academic and commercial licenses are available, as well as 30-day evaluation license for non-academic organizations.

4. General purpose

   Möbius is a software tool for modeling the behavior of complex systems. Although it was originally developed for studying the reliability, availability, and performance of computer and network systems, it is now used for a broader range of discrete-event systems, from biochemical reactions within genes to the effects of malicious attackers on secure computer systems, in addition to the original applications.

5. Platform: Windows, Linux Fedora Core 3 and newer. Java runtime environment and C++ compiler are required.

6. Model

   (a) Model class: quantitative, analytical and simulation; IT-level.

   (b) Model type

   Möbius supports many different atomic model types, which can be mutually combined. Originally, it supported only stochastic activity networks (SAN), however in the meantime it supports standard modeling paradigms such as Petri nets or queues, stochastic process algebra, stochastic automata networks, fault trees and reliability block diagrams.

   (c) Model description

   The above model types can be combined and hierarchically related, as Möbius starts with the assumption that no single modeling language can be optimal for all problem domains. Therefore, an open platform is offered which integrates different modeling formalisms. Furthermore, models can be solved using state-based analytical / numerical methods or using discrete events simulation. Note that we provide a comprehensive coverage of the stochastic activity networks as well as modeling example using Möbius in the Section 3.1.6 and in the Appendix A.

   (d) Systems modeled: fault-tolerant systems, discrete-event systems such as biochemical reactions or computer networks.

   (e) Model input: model-dependent. For state-space models, list of states and transitions probabilities are given; for combinatorial model, parameterized elements and their topology.

   (f) Model output: reliability, availability, performability and security.

   (g) Interfaces

   Möbius offers a graphical user interface (model editor) for specifying and connecting different models. External atomic formalisms necessary for the specification/analysis can be added.

7. Use cases

   Möbius supports the validation of systems in multiple application domains, including:

   - Information technology systems and networks
   - Wired and wireless telecommunication software and hardware systems
   - Aerospace and aeronautical systems
   - Commercial and government secure information systems and networks
   - Biological systems

8. Assumptions and restrictions: only exponential and instantaneous transitions are allowed in state-space models.

B.1.25 NFTAPE

1. Source/Reference: [192], [191], [190]

2. Project status

   The project is being developed at the University-of-Illinois-at-Urbana-Champaign. The last activity is from year 2002. NFTAPE is the successor of the FTAPE project.

3. License type: academic license is available.

4. General purpose

   NFTAPE (Networked Fault Tolerance and Performance Evaluator) is a software environment for automated reliability assessment of a network using the fault injection method. With NFTAPE it is possible to define a fault injection plan, execute experiments based on the plan and finally to analyse the obtained experimental results. Fault injection process is described and activated in runtime using a script language. NFTAPE offers the following metrics: reliability, availability and coverage. It also supports evaluation of distributed systems.

5. Platform: Solaris, Linux

6. Model

   (a) Model class: quantitative – benchmarking and test; IT-level.

   (b) Model type: software-based fault injection.

   (c) Model description

   NFTAPE is based on a component architecture comprising of the following modules: Lightweight Fault Injector (LWI), Lightweight Fault Trigger (LFT) and a control environment. Using a debugger or device drivers faults can be injected into memory (main memory, registers, etc.), operating system, I/O devices and network adapters and controllers of the target (evaluated) platform. A process manager runs on the target platform, and receives and implements fault scenarios from the control environment.

   (d) Systems modeled: distributed (network) systems.

   (e) Model input: fault-injection scenarios.

   (f) Model output: reliability, availability, coverage and repair.

   (g) Interfaces

   - Input: language for fault injection scenario, implemented using Python scripts (Jython).
   - Output: graphical interface.

7. Use cases

   The known use cases of NFTAPE include:

- Motorola IDEN MicroLite: testing of a base station controller
- DHCP (Dynamic Host Configuration Protocol): assessment of a control flow of DHCP applications
- SIFT-Environment (Software Implemented Fault Tolerance) for the Remote Exploration and Experimentation project (REE)
- Server-based internet applications: evaluation of SSH- and FTP-services using fault injection

8. Assumptions and restrictions

   The tool does not offer an abstract fault-injection planing language (Python is used instead). No source code modification or scaling is possible.

B.1.26 NUMAS

1. Source/Reference: [151], [20]
2. Project status: the last known developments are dated 1989. The project is not active.
3. License type: N/A.
4. General purpose

   NUMAS is a performance analysis tool. It is possible to extend each queueing station with fault-tolerance characteristic, hence fault-tolerant assessment is also possible using the queuing models.

5. Platform: Solaris.
6. Model
   (a) Model class: analytical, IT level.
   (b) Model type: queueing networks, Markov chains.
   (c) Model description

   A system being modeled is represented as a queuing network. It is then translated into Markov chain which is numerically solved by computing steady state distribution using Gaussian elimination, Gauss-Siedel iteration and iterative aggregation.

   (d) Systems modeled: repairable and non-repairable systems.
   (e) Model input: multi-server queues and the load dependent failure rates for servers; distribution of customers over the queuing stations; degradation mode of each queuing station.
   (f) Model output: steady-state performability and steady-state availability.
   (g) Interfaces

- Input: the NUMAS functionality can be accessed via two user interfaces: interactive NUMAS interface and HI-SLANG modeling language (HIT System Language). Both enable the user to input high-level degradable queuing network models. When using HI-SLANG, hierarchical modeling is also available (Markov chains within queuing networks).
- Output: steady state performability in tabular form, sorted according to input elements.

7. Use cases: N/A.

8. Assumptions and restrictions

   It is not possible to model dependencies in the failure or repair process of the different queuing stations. Only steady state measurement can be computed.

B.1.27 OpenSESAME

1. Source/Reference: [215]

   Web: http://www.lrr.in.tum.de/~walterm/opensesame/

2. Project status: the project is active, the last known activity is from 2007.

3. License type: academical license for research and teaching, commercial license is also available per request.

4. General purpose

   OpenSESAME offers a user-friendly modeling language which combines the simplicity of combinatorial availability models such as reliability block diagrams with the expressive power of the state-space based modeling techniques such as Markov chains. That means that mutual dependency between components can be specified.

5. Platform: Linux/Unix.

6. Model

   (a) Model class: quantitative, analytical; IT-level.

   (b) Model type: reliability block diagrams, failure dependency diagrams, stochastic Petri nets.

   (c) Model description

   A user inputs the model of a fault-tolerant system using RBD notation. The model is then internally transformed into a Petri net and analyzed using the DSPN-express-NG solver, which is built into OpenSESAME.

   (d) Systems modeled: redundant, fault-tolerant systems.

(e) Model input

The reliability block diagram of the system that should be analyzed is given as input. The elements are parameterized with MTTF and MTTR. Contrary to classic RBD notation, dependencies between RBD elements can be defined. Furthermore, it is possible to specify redundancy schemes, repair strategies and fault dependencies.

(f) Model output

Failures with common cause, fault prediction (destruction or malfunction/blocking of components), fault diagnosis, non-zero failover times for redundant systems in standby mode, limited repair capacities.

(g) Interfaces

- Input: graphical, tabular.
- Output: graphical, file.

7. Use cases

Known use-cases are modeling of Web servers, telecommunication systems, networks, power grids.

8. Assumptions and restrictions

The classical combinatorial assessment methods for RBD/FT cannot be applied.

B.1.28 PENELOPE

1. Source/Reference: [67], [66], [142], [221]

2. Project status: the last user-guide is from 1996, and the last publication from 1997. The current project status is unknown.

3. License type: academic license.

4. General purpose

PENELOPE is a software tool for performance analysis and optimization. It is based on the extended Markov reward model (EMRM), which extends Markov models with stationary optimization algorithms and different performance analysis strategies.

5. Platform: Sun OS 4/5.

6. Model

(a) Model class: quantitative, analytical/simulation; IT-level.

(b) Model type: extended Markov reward model (EMRM), controlled stochastic Petri nets (COSTPN), optimization and simulation.

(c) Model description
EMRM is an extension of Markov reward models, supporting reconfiguration arcs and branching states. COSTPN is an extension of the GSPN model, supporting additional transitions (e.g., reconfiguration). For numerical evaluation, COSTPN models are translated into EMRM.

(d) Systems modeled: dynamically reconfigurable systems.

(e) Model input: for EMRM, Markov models with extended arcs and states; for COSTPN, places, transitions, tokens and additional transitions.

(f) Model output: transient performance optimization, stationary performance optimization, performance simulation, transient performance analysis, stationary performance analysis.

(g) Interfaces
- Input: C-implementation, graphical user interface.
- Output: C-implementation, file and graphical.

7. Use cases: optimal strategy of an emergency supply.

8. Assumptions and restrictions: mission times have to be finite.

B.1.29 PENPET

1. Source/Reference: [127], [98]

2. Project status: the tool was developed by Siemens AG in cooperation with the University Mulhouse. Last publication is from 1996, the current project status is unknown.

3. License type: N/A.

4. General purpose: analysis of performability and reliability of fault-tolerant systems.

5. Platform: C-implementation for Unix (X-Window with MOTIF).

6. Model

(a) Model class: quantitative-analytical, qualitative; IT-level.

(b) Model type: generalized stochastic Petri nets (GSPN), Markov reward models.

(c) Model description
PENPET allows for high-level models to be described as clusters of lower-level models. Model reliability is described with a structural formula at the top level. Models called macro-molecules are defined, which comprise of sub-models called molecule-clusters or components called atoms. The textual description using structural formulas is

similar to the chemical description of molecules (hence the names). For each sub-component (molecule-cluster or atom), the quantity and the relationship with the upper level is specified. Failure rate and coverage are also specified. This completes the structural formula. The sub-models and components from the structural formula are then described using generalized stochastic Petri nets (GSPN), which are available in the library. After that, an automatic transformation translates the entire structural formula into a GSPN. Using this system GSPN, system states and underlying Markov chains are derived. Performance part is modeled using a reward model. Failure rates from the structural model are also included.

(d) Systems modeled: repairable and non-repairable fault-tolerant systems.

(e) Model input: structural formula (model) of a system; GSPN models for the components and sub-models (specified manually if no appropriate model can be found in the system library).

(f) Model output: instantaneous and steady-state performability and reliability.

(g) Interfaces

- Input: textual.
- Output: tabular or graphical.

7. Use cases: PENPET has been used for performability analysis of fault-tolerant multiprocessor systems.

8. Assumptions and restrictions: N/A.

B.1.30 Relex Reliability Studio: PRISM

1. Source/Reference: N/A.

   Web: http://www.relex.com

2. Project status: active.

3. License type: commercial license.

4. General purpose

   Reliability Studio is a software toolkit produced by the Relex company, which offers reliability, performance and maintenance assessment of computer systems. It supports methods and processes for failure criticality evaluation as well as models for reliability and availability analysis. Using simulation and optimization different statistical assertions about these criteria can be derived. PRISM is a standard for reliability analysis and MTBF prediction, developed by Reliability Assessment Center (RAC). PRISM-tool is integrated into Reliability Studio, supporting other modules such as RBD, FT, FMEA etc.

5. Platform: Windows.

6. Model

   (a) Model class: quantitative – analytical/simulation, qualitative; IT- and process-level.

   (b) Model type: Markov models, reliability block diagrams with Monte Carlo simulation, FMEA/FMECA, fault trees, Human-Factor-Risk analysis.

   (c) Model description

   Reliability Studio uses combinatorial (RBD, FT) or state space-based models for system specification. Apart from them, it supports risk analysis (FMEA, FMECA, HF-PFMEA, fault-trees) and process control (FRACAS, reliability prediction).

   (d) Systems modeled: repairable, non-repairable systems and redundant systems.

   (e) Model input

   - Markov models: functional dependencies using state transition diagrams
   - Risk analysis: process description
   - FMEA-FMECA: workflow

   (f) Model output

   The tool output is model-dependent, includes reliability, availability, unavailability, MTBF, MTTF, failure rates, expected number of failures, downtime, failure probability, hazard rate. More precisely the following outputs are supported by particular models:

   - Markov models: transient and steady-state availability, capacity, failure probability, cost and number of visits for every state in the Markov chain
   - RBD: steady-state and transient availability, confidence intervals
   - FMEA-FMECA: criticality matrix and order, risk level, risk priority
   - Risk analysis: risk assessment calculation

   (g) Interfaces: Web-based, graphical and tabular (Excel, Access, text); reporting available in Excel, PDF and HTML formats.

7. Use cases are known in the following industrial sectors: aerospace, automotive, oil, medical devices, railway, telecommunications.

8. Assumptions and restrictions: N/A.

B.1.31 QUAKE

1. Source/Reference: [200]

2. Project status

   The tool was developed at the University Coimbra, Portugal together with INRIA, France and is still in active development.

3. License type: N/A.

4. General purpose

   QUAKE is a tool for availability assessment of grids and Web services.

5. Platform: N/A.

6. Model

   (a) Model class: quantitative – benchmarking and test; IT- and service-level.

   (b) Model type: fault-injection, time series analysis, benchmark.

   (c) Model description

   QUAKE consists of the Benchmark Management System (BMS) and the SOAP Webserver called System Under Test (SUT). BSM comprises modules for automatic benchmark execution (definition, execution and result persistence).

   The Webserver is operated under defined workload and faultload. Different SOAP-XML client requests are simulated, and all grid nodes are synchronized via the Network Time Protocol (NTP). Besides SOAP, other middleware protocols are supported.

   The benchmarking supports different modes: in the learning mode, basic performance data are collected. In the workload mode the server is tested under the increasing workload (stress-testing). Additionally, server behavior can be influenced using fault injection into the system resources (this is the workload/faultload mode).

   The workload can be generated using following distributions: continuous maximal workload, stationary distribution, maximal time-limited workload, and surge load (constant request number with occasional peaks).

   The faultload in induced using an external program. The following system resources can be tested: main memory usage, increased number of threads, extensive use of file descriptors, usage of database connections.

   (d) Systems modeled: distributed, Web-based systems (Web Services, grid nodes).

   (e) Model input

   The rule-based language exists which enables description and configuration of procedures for workload and faultload scenarios. More details can be found in [200].

   (f) Model output

- Failures, their characterization (e.g., hung-up, crash, zombie-server), client-perceived failures (log files)
- Basic performance metrics: average number of requests per second, average response time for a request, conforming requests, service functionality, turn-around time
- Performance under varying workload distributions and additionally, with fault injection at the server side
- Reliability metrics:
  - Integrity: number of Web service faults under increased workload and faultload
  - Availability
  - Autonomy: number of cases where manual intervention is required (hang-up), as opposed to crash where the server can reboot itself automatically
  - Self-healing effectiveness

(g) Interfaces: textual form (input), numerical and tabular form (output). More details are not disclosed.

7. Use cases: performance and stress testing of grids and their services.

8. Assumptions and restrictions: N/A.

B.1.32 Reliability Center: PROACT, LEAP

1. Source/Reference: N/A.

   Web: http://www.reliability.com

2. Project status: the project is active.

3. License type: commercial license.

4. General purpose

   PROACT (Preserving Event Data, Ordering the Analysis Team, Analyzing Event Data, Communicating Findings & Recommendations Tracking For the Bottom-Line Results) is the name of the software product for the Proact process model, which is a variation of the Root Cause Analysis method (RCA). PROACT enables to establish this process in a company according to the given rules. LEAP implements the FMEA method (Failure Modes and Effects Analysis).

5. Platform: Windows.

6. Model

   (a) Model class: quantitative - analytical, qualitative; IT-level.

   (b) Model type: PROACT uses process model following the Root-Cause-Analysis method. LEAP uses FMEA and opportunity analysis models.

(c) Model description

The RCA method belongs to the analytical methods. Using the logic tree diagram, the causes and their effects are modeled and investigated. The RCA logic tree supports process, technical infrastructure and personnel availability. The FMEA method is a qualitative method for analysis of potential failure modes within a system for classification by severity or determination of the effect of failures on the system.

(d) Systems modeled: technical systems (PROACT) and organization structures (LEAP).

(e) Model input

- PROACT: description of process causes (physical and personnel related), as well as their effects.
- LEAP: probabilities and historical data.

(f) Model output

- PROACT: data about the process effectivity, recommendations.
- LEAP: ranking of undesirable events.

(g) Interfaces: textual form (input); graphical, online, tabular (output).

7. Use cases

The known use cases are in the sectors of air and space travel (NASA), railway (AMTRAK), whole food production, electrical engineering (General Electric), automotive (General Motors), power industry (Virginia Powers), oil and gas (Shell).

8. Assumptions and restrictions: N/A.

B.1.33 Reliass

1. Source/Reference: N/A.

Web: http://www.reliability-safety-software.com

2. Project status: the project is active.

3. License type: commercial license, demonstration version is also available.

4. General purpose

Reliass is used for reliability and security analysis. It offers the following modules:

- ASENT Toolkit: reliability and maintainability analysis
- AIMSS: management of technical information about the complex systems
- Logan Fault Tree: fault tree and event tree analysis
- Raptor: reliability analysis using reliability block diagrams

- RAMP: simulation of process-based systems
- RAM Commander: toolkit for reliability analysis
- D-LCC: calculation of lifecycle costs
- MEADEP: data-based reliability analysis
- FavoWeb: tool that uses FRACAS method
- FMEA-Pro6: implementation of the FMEA method
- PHA-Pro6: tool that implements hazard analysis
- PRISM: toolbox of procedures for reliability prediction

5. Platform: Windows.

6. Model

   (a) Model class: quantitative – analytical and simulation, qualitative; IT-level.

   (b) Model type
   The following models and methods are supported:
   - FMECA, RCM and testability analysis (ASENT, RAM Commander)
   - fault preventing maintenance, thermal analysis, sensitivity analysis (ASENT)
   - RBD (ASENT, Raptor) with Monte-Carlo simulation (RAM Commander), hierarchical RBD/Markov models (MEADEP)
   - fault tree and event tree models (Logan Fault Tree, RAM Commander)
   - Weibull failure distribution (RAMP)
   - Reliability prediction (RAM Commander)
   - Weak-link-analysis and phased simulation models (Raptor)

   (c) Model description
   Apart from basic reliability models (RBD, FT, Markov), the RAM Commander offers RBD together with Monte-Carlo simulation module. This is RBD extension for cases where analytical solution is not possible to calculate, such as stand-by, active redundancy, mutually dependent elements or limited repair capabilities configurations. Furthermore, MEADEP offers a framework for the historical analysis of reliability data. It consists of the following modules: data preparation, data editor and analysis, model generator and model solver. The results are derived either directly from the underlying data, or from the reliability models. In the former case, the data must be preprocessed, imported and then modeled using e.g., RBD or Markov models.

   (d) Systems modeled: repairable, redundant and complex systems (mixed serial/parallel and complex configurations).

   (e) Model input: system model, reliability model, system parameters.

(f) Model output: numerical reliability metrics including MTBCF, MTTR, Man-hours, reliability, availability.

(g) Interfaces

- Input: graphical user interface is provided for data and model input. Furthermore, MEADEP supports data import from flat ASCII files and databases.
- Output: graphical and tabular output is supported, as well as file export. Results can be exported graphically as bitmaps, or as alphanumerics into databases and popular Office products (e.g., Excel, Word).

7. Use cases: air and space industry, powerplants, energy sector, integrated circuits design.

8. Assumptions and restrictions: N/A.

B.1.34 Reliasoft

1. Source/Reference: N/A.

   Web: http://www.reliasoft.com

2. Project status: the project is active.

3. License type: commercial and free evaluation licenses are available. Furthermore, there are several commercial license types: single user, standard and concurrent network, unlimited and lease license.

4. General purpose

   The tool performs reliability analysis and offers the following packages:

   - Weibull++: Life data analysis (Weibull analysis)
   - Accelerated Life Testing Data Analysis (ALTA): analysis of data obtained during stress tests
   - Reliability Growth and Repairable Systems Data Analysis (RGA): reliability growth analysis software with fielded (repairable) system analysis capabilities for determining the optimum overhaul time
   - BlockSim: system reliability and maintainability analysis utilizing a reliability block diagram (RBD) or fault tree analysis (FTA) approach to obtain system results based on component data
   - Xfmea: Failure Modes, Effects and Criticality Analysis (FMEA/ FMECA) component
   - RCM++: analysis, data management and reporting for Reliability Centered Maintenance (RCM) analysis, integrated with FMEA/FMECA capabilities
   - Lambda predict: supports standards for reliability prediction analysis, including MIL-217, Bellcore, NSWC-98, China 299B and RDF 2000

- Reno: visual simulation software for risk and decision analysis
- XFRACAS: incident (failure) reporting, analysis and corrective action software system based on the FRACAS method

5. Platform: Windows.

6. Model

   (a) Model class: qualitative and quantitative, analytical and simulation; IT-level.

   (b) Model type: life data and lifetime distributions (Weibull++), life-stress relationships (ALTA), reliability growth analysis models (such as Crow-AMSAA, Duane, Standard Gompertz, Lloyd-Lipow, Modified Gompertz and Logistic in RGA), reliability block diagrams and fault trees (BlockSim), flowcharts and stochastic models (Reno), FMEA / FMECA standards (such as AIAG FMEA-3, J1739, ARP5580, MIL-STD-1629A in Xfmea), RCM models (such as MSG-3 and SAE JA1012 in RCM++), FRACAS models (in XFRACAS).

   (c) Model description

   Weibull analysis performs life data analysis utilizing multiple lifetime distributions, such as 1-, 2- and 3-parameter Weibull, mixed Weibull, 1- and 2-parameter exponential, lognormal, normal, generalized Gamma, Gamma, logistic, loglogistic, Gumbel and Weibull-Bayesian lifetime distributions. Life data can be also used for quantitative accelerated life testing data analysis. Reliability growth models are used to analyze time-to-failure (continuous), success/failure (discrete) and reliability data from various types of developmental (reliability growth) tests. Different types of developmental (reliability growth) testing strategies can be employed: test-fix-test, test-find-test or test-fix-find-test. This methodology enables reliability growth projections and provides a method to evaluate the reliability growth management strategy. RBD and FT are used for the standard reliability analysis. Flowcharts can express ordering of stochastic processing, which can be subsequently simulated. Reliability prediction models are used when actual product reliability data is not available. In such cases, standards based reliability prediction (MIL-217, Bellcore, NSWC, RDF 2000 and China 299B) may be used to evaluate design feasibility, compare design alternatives, identify potential failure areas, trade-off system design factors and track reliability improvement. A failure modes and effects analysis (FMEA) is a procedure for analysis of potential failure modes within a system for classification by severity or determination of the effect of failures on the system. Failure causes are any errors or defects in process, design, or item, especially those that affect the customer, and can be potential or actual. Effects analysis refers to studying the consequences of those failures. RCM models are based

on the equipment selection, failure effect categorization and maintenance task selection. They allow to compare possible maintenance strategies based on cost and/or availability estimates obtained by simulating the equipment's operation. Finally, XFRACAS models are designed for the acquisition, management and analysis of product reliability, quality and safety data from multiple sources, along with the management of problem resolution activities.

(d) Systems modeled: physical (technical), economic and organizational systems.

(e) Model input:

- life data
- stress test results
- RGA model configuration
- parameterized reliability block diagram or fault tree
- stochastic processes connected in a flowchart description
- system configuration, component characteristics and operating conditions
- FMEA worksheet, hierarchical tree or filtered list
- Failure Effect Categorization (FEC) and Maintenance Task Selection logic charts
- incident management process configuration

(f) Model output:

- warranty projections, degradation analysis, confidence bounds
- accelerated life testing time-to-failure, use-level cumulative distribution function, reliability, probability of failure, warranty time, mean life
- MTBF, reliability or failure intensity given time (cumulative or instantaneous), expected number of failures given time, time/stage to achieve a given MTBF or failure intensity, demonstrated MTBF or failure intensity, projected MTBF or failure intensity, maximum growth potential, unseen failure modes
- reliability, maintainability, availability, throughput, life cycle cost, failure rate, MTTF, warranty time, reliability importance plot, cost-effective component reliability allocation strategy, maintenance duration and restoration factors, maintenance policies
- risk and safety analysis, decision plan, maintenance plan
- failure rates, MTBF, Pi-Factors
- FMEA criteria and classifications, projections, item properties, sorted failures, effects, causes and controls by description, recommended actions
- optimum maintenance interval, operational costs of various maintenance strategies, risk priority numbers

(g) Interfaces

- Input: all tools provide graphical editors.
- Output: extensive reporting facilities are provided (graphical, file-export, databases, Office).

7. Use cases: aerospace, automotive, chemical and process, oil and gas, defense, energy, electronics, medical/health care, telecommunications, semiconductor, transportation, IT hardware.

8. Assumptions and restrictions: N/A.

B.1.35 SAVE

1. Source/Reference: [63], [90], [91], [92]

2. Project status

   SAVE is based on ARIES, CARE III and HARP tools. Project is currently inactive, but Monte Carlo techniques it employs are used in a number of commercially available tools (see Figures 4.2 and 4.3).

3. License type: N/A.

4. General purpose: solving probabilistic models of system availability and reliability of mission-oriented and continuously operating systems.

5. Platform: FORTRAN 77.

6. Model

   (a) Model class: analytical and simulation, IT level.

   (b) Model type: homogeneous Markov chain.

   (c) Model description

   Steady-state availability is computed by solving the homogeneous set of simultaneous linear equations derived from the Markov chain. Sensitivity with respect to the transition rate parameters (failure and repair rate) is calculated by differentiating the linear equations that satisfy the Markov chain. Mean time to failure is obtained from the transient behavior of the system. Models can be solved also by simulation using direct and analog Monte Carlo methods.

   (d) Systems modeled: repairable and non-repairable systems.

   (e) Model input

   The models supports following input parameters:

   - Method to be used for solving: numerical, Markov, combinatorial
   - Components specified by type (e.g., processor, database, spare)
   - For each component, a spare is defined along with the spare's failure rate
   - Operational dependencies for components

- Failure rates for components (in dormant and operational state)
- Repair rates for components
- Repair dependencies
- Failure modes for the system
- Failure mode probabilities for each component
- For each failure mode, affected components are specified
- The conditions under which the system is considered operational are specified
- Repair strategy (order in which the components are repaired)

(f) Model output

The following outputs can be calculated/simulated by the solver:

- Steady-state availability
- Sensitivity analysis
- Mean time to failure

(g) Interfaces

- Input: batch language is used for specifying model inputs.
- Output: results are written into a file, therefore, a file system interface (handler) is enough to communicate with the tool

7. Use cases: space computers and avionics (non-repairable systems) and telephone switching systems, general purpose computer systems, transaction processing systems (repairable systems).

8. Assumptions and restrictions

The tool works with exponential distributions only. Furthermore, it does not address transient and intermittent faults and is not able to compute transient (instantaneous) availability.

B.1.36 SHARPE

1. Source/Reference: [171], [172], [103], [167], [202], [170], [201], [173]

Web: http://people.ee.duke.edu/~kst/software_packages.html

2. Project status

First version of the SHARPE tool was based on the SPADE, DEEP and HARP tools. The tool has evolved and currently exists under the SHARPE 2000 and SHARPE 2002 names, also featuring Web extensions. The project is active and under continuous development.

3. License type

SHARPE 2000/2002 is free for academic use (see http://shannon.ee.duke.edu/tools/agreement_sharpe.htm). A commercial license for SHARPE 2000/2002 exists, but its conditions (e.g., price, duration, number of users) are not disclosed.

4. General purpose

   SHARPE 2000/2002 is a toolkit that provides a specification language and solution methods for most of the commonly used model types for performance, reliability and performability modeling.

5. Platform: Windows, Linux, Solaris; The tool is Java-based and requires/can be executed in any available JVM.

6. Model

   (a) Model class: analytical and simulation, IT level.

   (b) Model type: Markov chains (irreducible, acyclic, phase-type), semi Markov chains, reliability block diagrams, fault trees, reliability graphs, single-chain product-form queueing networks, multiple-chain product-form queueing networks, generalized stochastic Petri nets, series-parallel graphs.

   (c) Model description

   The tool enables specifying and analyzing performance, reliability and performability models. Model types include combinatorial, such as fault-trees and queuing networks, and state-space, such as Markov and semi-Markov reward models as well stochastic Petri nets. Steady-state, transient and interval measures can be computed. Output measures of a model can be used as parameters of other models. This facilitates the hierarchical combination of different model types. Overview of models and capabilities is given in table B.1.

	reliability	performance	performability
fault tree	X		
multistate fault tree	X		
reliability block diagram	X		
reliability graph	X		
Markov chain	X	X	X
semi-Markov chain	X	X	X
markov regenerative process	X	X	X
generalized stochastic Petri net	X	X	X
Stochastic reward net	X	X	X
product-form queuing network		X	
multi-chain queuing network		X	
task graph		X	
phased-mission systems	X		

Table B.1: Sharpe model description

   (d) Systems modeled: non-repairable and repairable systems.

   (e) Model input

   Model input varies depending on the used modeling language:

- Markov chains with absorbing states (acyclic or PH-type): transitions and transition rates; rewards; initial state probabilities. Irreducible Markov chains: transitions and transition rates; rewards; initial state probabilities (for transient analysis only).
- Acyclic semi-Markov chains: transitions and transition distributions; rewards; initial state probabilities.
- Irreducible semi-Markov chain: transitions and transition distributions; rewards.
- Reliability block diagram input syntax is: `block name {(param_list)} <blockline> end`. A blockline can further specify: component (name and exponential polynomial); parallel combination of components; serial combination of components; k-out-of-n system having identical components; k-out-of-n system having different components.
- Similar to RBD, fault trees input syntax is: `ftree name {(param_list)} <ftreeline> end`. A ftreeline can further specify: distinct event type with assigned exponential polynomial; repeating event; transferred events; and gate; or gate; k-out-of-n gate with identical inputs; k-out-of-n gate with different inputs.
- Reliability graphs: unidirectional edges (names and exponential polynomial cumulative distribution function for the time-to-failure of the path); bidirectional edges (names and exponential polynomial cumulative distribution function for the time-to-failure of both paths).
- Single-chain product-form queuing networks: station-to-station probabilities; station types and parameters; numbers of customers per chain.
- Multiple-chain product-form queueing networks: station-to-station probabilities for each chain; station types and parameters; number of customers per chain.
- Generalized stochastic Petri nets: places and initial numbers of tokens; timed transition names, types and rates; immediate transition names and weights; place-to-transition arcs and multiplicity; inhibitor arcs and multiplicity.
- Series-parallel graphs: edges; exponential polynomial for each graph node; exit types for given node (parallel subgraphs are probabilistic; all of the parallel subgraphs must complete; one of the parallel subgraphs must complete; k-out-of-n parallel subgraphs must complete); probabilities for edges; multipath information.

(f) Model output

In general, the following outputs are given, at various model levels: reliability of selected components, system reliability over an interval, system steady-state availability, transient availability. The detailed output is given below:

- Cumulative distribution function
- Series-parallel acyclic directed graph: graph execution time; if multipath information is requested, cumulative distribution function is given for each path.
- Reliability block diagram, fault tree, reliability graph: if the function assigned to each component is the cumulative distribution function of its failure time, then output is system failure time cumulative distribution function. If the function assigned to each component is the instantaneous or steady-state availability, output is system instantaneous or steady-state availability.
- Acyclic semi-Markov chain, acyclic and phase-type Markov chain: if no state name is given, cumulative distribution function is given for the time until some absorbing state is reached. If state is given, output is distribution function for the time until the state is reached and the probability of the state being reached. If the transient state is reached, output is transient probability function of being in that state.
- Irreducible Markov chain: transient probability function of being in a given state.
- Non-irreducible generalized stochastic Petri nets: time until reaching an absorbing marking.
- Conditional probability that the state has been left by given time, mean and variance of the cumulative distribution function, probability of visiting a state.
- Markov and semi-Markov models with rewarding states: Probability that the accumulated reward at the time of absorption is less or equal to the given value
- Generalized stochastic Petri nets: the average number of tokens in the specified place, probability that the place is empty, utilization and throughput of a transition (steady states, at given time symbolic and numeric, time-averaged).
- Product-form queuing networks: throughput, average response time, average queue length, utilization (single- and multi-chain networks).

(g) Interfaces

- Input
  SHARPE 2000/2002 accepts inputs using SHARPE batch language or Java-based graphical user interface (GUI). Batch language has defined syntax and semantics and allows for creation, modification, and analysis of hierarchical models. GUI supports the same options. Programmable external input connections with SHARPE are therefore best established using batch language, although it can be speculated that elements of Swing user interface can also be accessed from external programs. A Web-based shell for SHARPE is also available.

- Output
  SHARPE 2000/2002 presents simulation results within Java-based GUI. Non-graphical results can be obtained using batch language, wherein results can be received as text files. Export of models and analysis results is supported within the GUI shell (e.g., to Excel, JPEG or EPS files). Aggregation of graphical results from multiple experiments is possible. A Web-based shell for SHARPE is also available.

7. Use cases: the project Website claims that package has been installed at 280 locations, however no published use cases were found up to this time.

8. Assumptions and restrictions: N/A.

B.1.37 SPNP

1. Source/Reference: [61]

   Web: http://people.ee.duke.edu/~kst/software_packages.html

2. Project status

   The software tool was developed at the Duke University, but last modifications are from the period 1989-2001. The project seems to have been partially incorporated into SHARPE, although the tool is still offered as a separate download.

3. License type: academic and commercial license.

4. General purpose: SPNP (Stochastic Petri Net Package) is a modeling tool for performance, reliability and performability analysis of complex fault-tolerant systems.

5. Platform: version 6 was compiled for MS-DOS, Solaris and Linux.

6. Model

   (a) Model class: quantitative-analytical, IT-level.

   (b) Model type: stochastic reward net (SRN), fluid stochastic Petri net (FSPN).

   (c) Model description

   The model type used for input is a stochastic reward net (SRN). An SRN incorporates several structural extensions to GSPN such as marking dependencies (marking dependent arc cardinalities, guards, etc.) and allows reward rates to be associated with each marking. The reward function can be marking dependent as well. SRN is specified using CSPL (C based SRN Language) which is an extension of the C programming language with additional constructs for describing the SRN models. SRN specifications are automatically converted into a Markov reward model which is then solved to compute a variety of transient, steady-state, cumulative, and sensitivity measures.

For SRN with absorbing markings, mean time to absorption and expected accumulated reward until absorption can be computed. In the latest version, fluid stochastic Petri net and non-Markov SPN can be also specified. They are solved using discrete-event simulation.

(d) Systems modeled: repairable and non-repairable systems.

(e) Model input

SRN specification is used as input. Additionally to usual SPN input elements (places, transitions, rates), SRN allows for marking-dependent arc cardinalities, guards and reward rates for markings, which can be also marking-dependent.

(f) Model output

The tool can calculate transient and steady-state reliability/ availability, as well as cumulative and sensitivity measures (average number of tokens, steady-state number of tokens, transition throughput). For SRN with absorbing markings, mean time to absorption and expected accumulated reward until absorption can also be computed.

(g) Interfaces

- Input: models are specified using CSPL (C based SRN Language) which is an extension of the C programming language with additional constructs for describing the SRN models. SRN specifications are automatically converted into a Markov reward model which is then solved.
- Output: output is written into a text file.

7. Use cases: software performance analysis, database availability, ATM network under overload, channel recovery scheme in a cellular network, performance analysis of MPLS network, reactor temperature control system, etc.

8. Assumptions and restrictions: N/A.

B.1.38 SoftRel LLC: FRESTIMATE

1. Source/Reference: [153], [154]

   Web: http://www.softrel.com

2. Project status: the project is active.

3. License type:

   The tool exists in three commercial versions: Frestimate Standard Edition, Frestimate Manager's Edition and Frestimate Metrics Package. Furthermore, there is an academic version called Frestimate evaluation / student edition, which is not free.

4. General purpose

Frestimate has been developed to help in the assessment of software reliability by computing the expected number of failures per 1000 lines of code. It is based on the monitoring of the project management and can be executed in a stepwise process by isolating critical production phases (e.g., lack of adequate testing). Furthermore, the cost-benefit analysis of each possible step can be performed, in order to quantify obtained benefits. It enables selection of those improvement steps of software reliability which offer the least complexity.

5. Platform: Windows.

6. Model

   (a) Model class: analytical/qualitative; IT-level, software.

   (b) Model type: correlation of software development practices. The result is a model to predict software reliability before the software is developed or tested. This model can also be used to determine the potential software process improvement areas to improve software reliability with the most balanced approach.

   (c) Model description

   The model is generated by the assessment of economic and organizational processes within the software development process. A correlation factor is then derived between the results thus obtained and the existing knowledge base. After that, based on the correlation factor, project (and software) reliability is computed. The following metrics can be extracted from the knowledge base: faults per 1000 lines of code, MTTF, reliability, availability, failure rate.

   (d) Systems modeled: software systems.

   (e) Model input

   A questionnaire is answered and the answers are used as the model input. Depending on the tool version, the questionnaire can have between 15 and 80 questions.

   (f) Model output

   The tool computes the following metrics for a software project:

   - An initial software reliability assessment
   - Predicted defects and defect density at start and end of testing as well as any time during useful life
   - Predicted failure rate, critical failure rate, MTTCF, MTTF, availability and reliability
   - Predicted staff required to fix residual defects each month after delivery
   - Predicted staff required to test the software to reach a desired level of defect density at delivery
   - Determining how software failure rates and components will be merged into the overall system reliability block diagram

All of the above predictions are completed for 4 milestones

- Start of test
- End of test (deployment)
- Average useful life
- End of useful life (point at which software version is replaced with next major software version)

(g) Interfaces

- Input: input is performed using a graphical user interface, where user is guided through a predefined set of questions.
- Output: graphical user interface, export to a file is supported, as well as interface with the database containing the tool knowledge base.

7. Use cases: software projects in the sector of semi-conductor design, defense, air and space construction.

8. Assumptions and restrictions

   It is assumed that the process in the assessed project is correlated/ compatible with the projects already contained in the knowledge base. However, the number of use-cases in the knowledge base is currently relatively small, and the precision of the method may therefore be questionable. This may especially be the case when no template is provided in the knowledge base (e.g., as is the case for the finance/banking software).

B.1.39 SURE

1. Source/Reference: [41], [42], [43]

   Web: http://shemesh.larc.nasa.gov/people/rwb/sure.html

2. Project status: the last change is from 2001, the project may be further developed internally within NASA.

3. License type: public domain, using special NASA public domain license.

4. General purpose

   SURE (Semi-markov Unreliability Range Evaluator) is a reliability analysis program used for calculating upper and lower bounds on the operational and death state probabilities for a large class of semi-Markov models.

5. Platform: Sun SPARC stations (SunOS), Linux, Windows

6. Model

   (a) Model class: analytical, IT-level.

   (b) Model type: semi-Markov models.

(c) Model description

Semi-markov Models are used to determine the bounds on operational and failure states. The calculated bounds are close enough (usually within 5 percent of each other) for use in reliability studies of ultra-reliable computer systems. The SURE bounding theorems have algebraic solutions and are consequently computationally efficient even for large and complex systems. The tool can optionally regard a specified parameter as a variable over a range of values, enabling an automatic sensitivity analysis.

(d) Systems modeled: fault-tolerant reconfigurable systems.

(e) Model input

Instead of using fault trees or a PMS description for input, an abstract language was created for defining the set of rules used to create the state transition matrix. All states in the system are first assigned a number, and then semi-Markov model is specified by enumerating all transitions. The SURE input language includes two statement types: model-definition statements and commands. Model specification is performed by enumerating transitions. It is allowed to use constants, variables, expressions (mathematical operations, standard functions), transition rates (fast or slow). SURE commands control the behavior of the SURE program, e.g., EXIT, INPUT, ECHO, PLOT or RUN. The language also supports loops and conditions.

The model definition part consists of five types of statements: the constant-definition statement, the SPACE statement, the START statement, the DEATHIF statement and the TRANTO statement. A constant-definition statement equates an identifier consisting of alphanumerics to a number. The SPACE statement specifies the state space on which the Markov model is defined as an n-dimensional vector, where each component of the vector defines an attribute of the system being modeled. The START statement indicates which state is the start state of the model that represents the initial state of the system. The DEATHIF statement specifies which states are death states (i.e., trapping states in the model). The TRANTO statement is used to generate all of the transitions required for a model recursively.

(f) Model output

Model solver offers the following outputs:

- Upper and lower bounds on the probability of total system failure
- Probability bounds for each death state in the model
- List of every path in the model and its probability of traversal

(g) Interfaces

- Input: models can be specified in a text (file) form, or using a GUI (WinSURE).

- Output: textual (file), GUI (export into image file), GNUPLOT (graph export).

7. Use cases: space industry (NASA).

8. Assumptions and restrictions

   Only reliability analysis is supported, repairable systems cannot be modeled. It is assumed that the nonexponential transitions must be faster when compared with the mission time. This is, however, a desirable attribute of all fault-tolerant systems.

B.1.40 SURF-2

1. Source/Reference: [78], [13], [114], [115]

   Web: http://www.laas.fr/surf/what-uk.html

2. Project status: the last publication as well as the project Website update is from 1996 and 1997 respectively. It can be assumed that the project has been discontinued.

3. License type: commercial license, as well as academic/education.

4. General purpose

   The SURF-2 is a dependability evaluation tool for hardware and software systems, based on construction, validation and numerical resolution of Markov models. System behavior is modeled by either a Markov chain or a generalized stochastic Petri net.

5. Platform: SUN-4, Sparcstation-4, Sparcstation-5, UltraSparc; SUN OS 4.1.x or SOLARIS 2.x; X-Window (X11R5).

6. Model

   (a) Model class: analytical, IT-level.

   (b) Model type: Markov chain, generalized stochastic Petri nets.

   (c) Model description

   System behavior is modeled by either a Markov chain or a generalized stochastic Petri net (GSPN). Reward structures can be added to the behavioral model and permit to get combined measures of dependability, performance or cost. Quantitative evaluation of system dependability can be divided in two steps:

   - Construction of the model describing the behavior of the studied system based on elementary stochastic processes corresponding to the behavior of the system components and of their interactions.
   - Mathematical processing of the model to get analytics expressions or numerical values of the system dependability measures.

Dependability measures are obtained from the processing of the Markov chain. The transformation of the GSPN into a Markov chain in continuous time is based on the markings which sensibilize timed transitions.

(d) Systems modeled: repairable and non-repairable systems.

(e) Model input

A GSPN or a Markov chain is specified using the following form:

- graph
- initial probability vector of the Markov chain or of the underlying Markov chain in case of GSPN
- one or several partitions, which define the characteristics of a system necessary to compute the dependability measures. It is made of:
  - a class of failure states named "IMPROPER",
  - a class of dangerous states named "CATASTROPHIC".

  These two state classes allow the definition of most of the dependability measures, "CATASTROPHIC" class being used for safety measures. For the same model, it is possible to define several partitions in order to analyze various failure cases.

Model parameters can be numerical or symbolic expressions. A symbolic parameter is a local variable which is visible only in the model in which it has been defined. The combined use of symbolic parameters and the definition of several partitions for a same model permits to build generic models which can be stored in the SURF-2 database in order to be reused.

(f) Model output: standard reliability and availability measures, as well as combined measures of dependability, safety, maintainability, performance or cost.

(g) Interfaces

- Input: SURF-2 offers two graphical model editors: Markov chain editor and Petri net editor.
- Output: results are displayed graphically and can be exported into the file in a tabular form.

7. Use cases: hardware/software co-design, air traffic control systems.

8. Assumptions and restrictions: N/A.

B.1.41 Sydvest CARA Fault Tree

1. Source/Reference: N/A.

   Web: http://www.sydvest.com/Products/Cara/prod_info.htm

2. Project status: commercial product, active.

3. License type: commercial license, with available evaluation/demonstration version offering limited functionality.

4. General purpose: the tool is specialized in the reliability and component importance analysis using fault trees.

5. Platform: Windows.

6. Model

   (a) Model class: analytical, IT-level.

   (b) Model type: fault tree analysis.

   (c) Model description

   The tool uses standard FTA method for analysing fault-tolerant behavior in the "fault" space.

   (d) Systems modeled: non-repairable systems.

   (e) Model input: the tool accepts a standard fault tree model.

   (f) Model output: minimal cut-sets, average availability, survival probability, MTTF, frequency of TOP event, failure frequency distribution, six measures of component importance (Birnbaum's reliability, Birnbaum's structural, Fussell-Vesely, criticality importance, improvement potential, order of smallest cut-set)

   (g) Interfaces

   - Input: text files formatted in accordance with producer's guidelines or a GUI. To improve the readability of large trees, the fault tree is structured into pages.
   - Output: GUI, it is further possible to export the reports as Rich Text Files (RTF) or to copy it on Windows clipboard.

7. Use cases

   The tool is used in Jardine Technology. It is a consulting company that helps its clients to resolve asset and risk management problems in the oil & gas, chemical, power, and transportation industries. However it is not clear to what extent has the tool been used or in which projects.

8. Assumptions and restrictions: the tool is applicable for fault tree based models and supports non-repairable systems only.

B.1.42 Sydvest Sabaton

1. Source/Reference: N/A.

   Web: http://www.sydvest.com/Products/Sabaton/

2. Project status: commercial product, active.

3. License type: commercial license, with available evaluation/demonstration version with limited functionality.

4. General purpose

   Sabaton supports FMEA (Failure Mode and Effects Analysis) and FMECA (Failure Mode, Effects and Criticality Analysis) which are typically used in product and system development to reveal possible failures and failure modes, and the effects of these failures. The analysis results typically in proposals for design improvement aimed at eliminating system failure or mitigating the effects of component failures.

5. Platform: Windows.

6. Model

   (a) Model class: analytical, IT- and service-level.

   (b) Model type: FMEA/CA

   (c) Model description

   A failure modes and effects analysis (FMEA) is a procedure for analysis of potential failure modes within a system for classification by severity or determination of the effect of failures on the system. It is widely used in manufacturing industries in various phases of the product life cycle and is now increasingly finding use in the service industry. Failure causes are any errors or defects in process, design, or item, especially those that affect the customer, and can be potential or actual. Effects analysis refers to studying the consequences of those failures.

   Failure Mode, Effects, and Criticality Analysis (FMECA) is an extension of Failure Mode and Effects Analysis (FMEA). In addition to the basic FMEA, it includes a criticality analysis, which is used to chart the probability of failure modes against the severity of their consequences. The result highlights failure modes with relatively high probability and severity of consequences, allowing remedial effort to be directed where it will produce the greatest value.

   (d) Systems modeled: N/A.

   (e) Model input

   Standard FMEA/ FMECA: failure mode, effects, severity rating, causes, occurrence rating, current controls, detection rating, critical characteristic, risk priority number (FMECA), recommended actions.

   (f) Model output

   Calculation of risk priority number (RPN) as well as other measures defined by the user. The RPN helps to assign priorities to potential failures and is used to rank potential design deficiencies and/or liability issues. Criticality analysis by use of criticality or risk matrix.

   (g) Interfaces

   - Input: text files formatted in accordance with producer's guidelines or through a GUI.

- Output: GUI that enables flexible reporting capabilities. It is possible to specify user definable analysis report templates. Reports can be saved in PDF format for electronic distribution.

7. Use cases: defense, health.
8. Assumptions and restrictions: N/A.

B.1.43 TANGRAM

1. Source/Reference: [21], [158], [101]
2. Project status: N/A.
3. License type: N/A.
4. General purpose

   TANGRAM is a general-purpose object oriented development environment, which can be customized to reliability and availability modeling, under assumption that adequate evaluation methods are available.
5. Platform: Unix, C implementation.
6. Model
   (a) Model class: depends on the application field.
   (b) Model type: depends on the application field.
   (c) Model description
   TANGRAM is a general purpose modeling environment, which can be customized for performability and reliability analysis, as well as for solving Markov reward models. Models are defined as collections of objects that are parameterized instances of object types (classes). Each object type has an internal state, which can be changed after internal events. Internal events can also cause sending of messages to other objects. Overall model state is computed as a composite state of internal object states. A reward can be associated with each object state. Because object types support inheritance, complex models may be constructed in a stepwise and hierarchical manner. It is possible, for example, to define high-level objects which can be reused and extended in multiple specialized models. This is the main advantage of TANGRAM.
   (d) Systems modeled: depends on the application field.
   (e) Model input: depends on the application field. In case of reliability/performability models, object states can represent functioning components and internal events may represent failures.
   (f) Model output depends on the application field. In case of reliability/performability models, evaluation of cumulative distribution functions describing events (such as failure) can be computed for an object or a model.

(g) Interfaces

The model input is performed using a text-based interface, compatible with the SAVE tool. Results can be queried using a custom query language.

7. Use cases: N/A.

8. Assumptions and restrictions: N/A.

B.1.44 Mathworks Stateflow

1. Source/Reference: N/A.

   Web: http://www.mathworks.com

2. Project status: commercial product in active development. The tool is frequently used together with other two Mathworks products, Simulink and Matlab.

3. License type: commercial license, academic license (reduced price), free 15-day evaluation and demonstration license.

4. General purpose

   Stateflow is a simulation tool for the event-based systems. Stateflow-charts enable graphical representation of the hierarchical and parallel states, as well as event-based deterministic transition between them. The tool extends statecharts with the following concepts: control flow, truth tables, temporal operations and event-based broadcasting. Stateflow is closely coupled with Matlab and Simulink which offer additional graphical modeling capabilities.

5. Platform: Windows, Linux, Solaris 8 or higher, MacOS.

6. Model

   (a) Model class: simulation, IT-level.

   (b) Model type: state charts, combined with finite state machines and temporal logic.

   (c) Model description

   A stateflow chart is a graphical representation of the finite state machine, which is defined in terms of states and transitions. Additionally, junctors and functional description (truth table, temporal logic) may be added. Three types of finite state machines are supported: Mealy, Moor and a hybrid Mealy-Moore machine type. The machines are constructed graphically. The charts support hierarchical modularization through a sub-chart concept. After specification, a statechart is evaluated through simulation.

   (d) Systems modeled: event-based fault-tolerant systems.

(e) Model input: functional and behavioral description of a system using an extended finite state machine model (connector for specifying control flow and truth tables or temporal logic expressions).

(f) Model output: system state before or after a specified event.

(g) Interfaces

The Matlab offers the following possibilities:

- definition of input/output events within the Stateflow models
- connecting of input/output ports with Simulink blocks
- definition of simulation parameters in Simulink

It is also possible to define an interface between the Stateflow block and external code. During the simulation it is possible to observe system states and their parameters.

7. Use cases: power and energy, avionics, automotive, telecommunications, space, defense.

8. Assumptions and restrictions: N/A.

B.2 Qualitative and Process Management Tools

B.2.1 Advanced Technology Institute: OCTAVE Automated Tool

1. Source/Reference: [64], [218]

   Web: http://www.aticorp.org/

2. Project status: N/A.

3. License type: commercial license, demonstration as well as a trial version is available for evaluation.

4. General purpose

   Octave Automated Tool has been implemented by Advanced Technology Institute (ATI) to help users with the implementation of the Octave and Octave-S approach. The tool assists the user during the data collection phase, organizes collected information and finally produces the study reports.

5. Platform: N/A.

6. Model

   (a) Model class: qualitative, IT- and process-level.

   (b) Model type: risk analysis.

(c) Model description

OCTAVE is a security risk evaluation tool. The core concept of OCTAVE is defined as a situation where people from an organization manage and direct an information security risk evaluation for their organization. They direct risk evaluation activities and are responsible for making decisions about the efforts to improve information security. In OCTAVE, an interdisciplinary team, called the analysis team, leads the evaluation.

In OCTAVE, risk has three aspects: organizational, technological, and analysis aspects. Therefore, OCTAVE is organized around these basic aspects and divided into following phases:

- Build asset-based threat profiles: this is an organizational evaluation. Staff members from the organization contribute their perspectives on what is important to the organization (information-related assets) and what is currently being done to protect those assets. The analysis team consolidates the information and selects the assets that are most important to the organization (critical assets). The team then describes security requirements for the critical assets and identifies threats to the critical assets, creating threat profiles.
- Identify infrastructure vulnerabilities: this is an evaluation of the information infrastructure. The analysis team identifies key information technology systems and components that are related to each critical asset. The team then examines the key components for weaknesses (technology vulnerabilities) that can lead to unauthorized action against critical assets.
- Develop security strategy and plans: during this part of the evaluation, the analysis team identifies risks to the critical assets and decides what to do about them. The team creates a protection strategy and mitigation plans to address the risks to the critical assets.

(d) Systems modeled: IT-organizations and processes.

(e) Model input

The input are process descriptions with the following attributes: analysis team, analysis team skills, catalog of practices, generic threat profile, catalog of vulnerabilities, defined evaluation activities, documented evaluation results, evaluation scope, next steps, focus on risk, focused activities, organizational and technological issues, business and information technology participation, senior management participation, collaborative approach.

(f) Model output

Each phase of the method produces a set of outputs:

- Phase 1 outputs: critical assets, security requirements for critical assets, threats to critical assets, current security practices, current organizational vulnerabilities

- Phase 2 outputs: key components, technology vulnerabilities
- Phase 3 outputs: risks to critical assets, risk measures, protection strategy, risk mitigation plans

(g) Interfaces

- Input: MS Access with Forms.
- Output: reporting into MS Word and Excel, as well as into Oracle database is supported.

7. Use cases: government, health, IT-organizations.

8. Assumptions and restrictions: N/A.

B.2.2 Alion Science and Technology: CounterMeasures

1. Source/Reference: N/A.

   Web: http://www.alionscience.com/index.cfm

2. Project status

   Alion is providing consulting service in the area of risk assessment for the US government, defense and commercial customers. The first version of CounterMeasures was made public in the 80s, the last commercial tool version is from 2007. The tool is in active development.

3. License type: commercial license with three licensing models (Enterprise, Standard and Web Survey), free evaluation license is also available.

4. General purpose

   CounterMeasures performs risk management based on the US-NIST 800 series and OMB Circular A-130 USA standards. The user standardizes the evaluation criteria and using a tailor-made assessment checklist, the software provides objective evaluation criteria for determining security posture and/or compliance.

5. Platform: Windows.

6. Model

   (a) Model class: qualitative, process-level.

   (b) Model type: risk analysis, separated into the identification, analysis and evaluation phases.

   (c) Model description

   The models enables semi-automated collection of risk relevant data, to improve efficiency but also to enable standardization and comparison. The repeatable process is used to analyze collected data which prioritizes risks, threats, and recommendations. The reports are generated afterwards that address vulnerability, threats, risk, and compliance. Finally, return on investment for certain recommendations (countermeasures) can be calculated. The tool also offers management of security/safety data in a repository (database).

(d) Systems modeled: physical systems, organizations, processes.

(e) Model input: process description, with the associated data collected using survey and data collection modules.

(f) Model output: risk report (description of risks associated with surveyed operations), remediation plan (plan of actions for facility security improvement), residual risk, risk treatment, risk acceptance and risk communication.

(g) Interfaces

- Input: MS Office/Forms-based questionnaire.
- Output: MS Excel, relational database export.

7. Use cases: government agencies, large scale companies in the sectors such as banks, gas/oil, insurance, ports, universities, states/municipalities, security.

8. Assumptions and restrictions: N/A.

B.2.3 Aprico Consultants: ClearPriority

1. Source/Reference: N/A.

   Web: http://www.aprico-consult.com/

2. Project status

3. License type: commercial license, can be scaled according to the project size.

4. General purpose

   The ClearPriority platform supports real-time data extraction, transformation and correlation capabilities, in order to enable enterprise risk assessment, tracking and reporting. The main part of the tool is security audit trail analyzer, which collects log file data across multiple systems, correlates these data and produces security alerts based on user defined rules. The user is able to define new sources of data as well as to specify the alert output.

5. Platform: platform-independent, any platform with Java Virtual Machine.

6. Model

   (a) Model class: qualitative, IT-level.

   (b) Model type: risk assessment through correlation rules and data transformation.

   (c) Model description

   The model allows for design and implementation of key performance, risk and compliance indicators across domains. After indicators have

been defined, data extraction and correlation rules, as well as statistical models used for reporting can be specified and implemented. Special support is given for the treatment (parsing and interpretation) of complex log (event and error) files. Rule-based detection of known threats processes risk events and enables real-time reporting and key performance indicators monitoring, as well as statistical and historical reporting.

(d) Systems modeled: complex software systems and their log files.

(e) Model input: key performance and risk indicators definition, raw log-file data.

(f) Model output: real-time indicator value, results of the rule-based risk assessment (risk analysis).

(g) Interfaces

- Input: Java- and XML-based interfaces, Risk ETL tool which collects raw audit trail information from any source (applications, operating systems, security devices and access control equipment producing audit logs, etc).
- Output: relational database interface (Oracle, SQL Server, Mysql) which can be queried using SQL.

7. Use cases: government, finance, education.

8. Assumptions and restrictions: N/A.

B.2.4 Aexis: RA2

1. Source/Reference: N/A.

   Web: http://www.aexis.de/RA2ToolPage.htm

2. Project status: the tool is actively developed, the first version was created in 2000, and the last major release was in 2005. In the period afterwards, the tool has been renamed into RA2 Art of Risk, and the last version is from 2008.

3. License type: commercial and evaluation licenses.

4. General purpose

   RA2 is a stand-alone tool for risk management based on the ISO 17799 and ISO 27001 standards. For each of the steps defined in these standards, the tool contains a dedicated step with report generation. RA2 Information Collection Device, a component that is distributed along with the tool, can be installed anywhere in the organization to collect and feed back information into the risk assessment process. The functions include leading through the information security management system (ISMS) processes, calculation of risks, automatic carrying forward and updating of results, a detailed help function and context sensitive help, and further support.

5. Platform: Windows.

6. Model

   (a) Model class: qualitative, IT- and process-level.

   (b) Model type: risk analysis and assessment based on ISO/IEC 17799 and 27001 standards.

   (c) Model description
   The tool provides software support to design and implement an information security management system (ISMS) in accordance with the requirements of ISO/IEC 27001 and 17799. This includes:
   - Defining the scope and business requirements, policy and objectives for the ISMS
   - Developing an ISMS asset inventory
   - Carrying out an ISMS risk assessment
   - Facilitating the risk decision process by consideration of the appropriate risk treatment option
   - Process for selecting a system of controls
   - Documentation facility for producing, for example, a Statement of Applicability and other ISMS documents

   (d) Systems modeled: IT-organizations and infrastructure.

   (e) Model input: asset inventory (an ISMS asset inventory, which can be selected from the example list or added as new), definition of the scope and business requirements policy and objectives for the ISMS, collected information from different sources within the organization.

   (f) Model output: risk assessment, suggested risk controls from ISO 17799, risk communication (report generator), compliance analysis results.

   (g) Interfaces
   - Input: Information Collection Device application, which collects information from different sources and provide to the tool.
   - Output: import/export (application specific): Information Collection Device, export to CSV, spreadsheet applications (e.g., Excel).

7. Use cases: security and risk consulting companies, large scale organizations and SMEs, government.

8. Assumptions and restrictions: N/A.

B.2.5 BMC: Remedy Suite

1. Source/Reference: N/A.
   Web: http://www.bmc.com/remedy/

2. Project status: commercial product, active.

3. License type: commercial.

4. General purpose

   The tool manages and automates ITIL processes in enterprise-level organizations thus improving project management quality throughout its lifetime.

5. Platform

   The suite supports Windows family, Unix and Linux operating systems. The Remedy IT Service Management (ITSM) can be integrated with network and systems management software, such as BMC PATROL, HP OpenView, Tivoli Enterprise Manager. In addition, it can be also integrated with business-critical systems, including enterprise resource planning (ERP) tools.

6. Model

   (a) Model class: qualitative, process-level.

   (b) Model type: subset of ITIL processes and COBIT control goals.

   (c) Model description

   Remedy IT Service Management (ITSM) software suite is certified as ITIL-compatible and provides support for various ITIL processes:

   - Incident Management
   - Configuration Management
   - Problem Management
   - Service Level Management
   - Change Management
   - Availability Management

   The producer claims that all BMC solutions apply to 27 of the 34 COBIT control objectives. CobiT coverage by IT domains is:

   - Plan and Organize: 6 of 10 control objectives
   - Acquire and Implement: 5 of 7 control objectives
   - Deliver and Support: 12 of 13 control objectives
   - Monitor and Evaluate: 4 of 4 control objectives

   A consistent definition of availability is not given, it is sometimes business oriented (making a customer support center "available" for extra hours and estimating its costs/benefits), on other occasions it is defined as whether a process in the system is running (which does not account for service availability one-to-one, a process might be still alive but not delivering the required service) or sometimes service availability is explicitly accounted but requires tight coupling and integration in the existing system (e.g., transactions in the SAP Enterprise Portal are used as an availability indicator).

   (d) Systems modeled: IT-organizations.

(e) Model input and output

Of particular importance for availability management and assessment is the Remedy Service Level Agreements tool which enables users to:

- Plan and manage availability of individual and grouped configuration items
- Monitor availability and performance to ensure that service level commitments are met internally (within organization) and externally. The monitoring can be cumulative time-based, event-based, and threshold-based.
- Use proactive alerts to identify issues and trigger actions prior to service level violations. For instance, Process Monitoring enables user to monitor process availability and resource consumption, and provides the ability to restart failed processes and terminate processes that consume a specified percentage of CPU time.
- Relate IT services performance to the service level agreements.

(f) Interfaces

Remedy ITSM suite architecture consists of a common database, workflow engine, reporting capabilities, and integrated development environment. The individual applications within the suite share a common workflow foundation and unified data model, thus supporting the integrated process approach outlined in the ITIL framework. The suite is large, consisting of 17 different tools, some of them heavily depending on other BMC products.

Remedy ITSM tools can be deployed as stand alone applications, in stages, and as the integrated application suite. Individual tools can be used out-of-the box or they can be adapted to fit the procedures and workflows used within a IT organization.

7. Use cases: telecommunications (TeliaSonera, Vodafone), semiconductors and electronics (Infineon), graphic and health (Agfa), hardware (Dell), etc.

8. Assumptions and restrictions: N/A.

B.2.6 BSI: GSTOOL

1. Source/Reference: N/A.

   Web: http://www.bsi.bund.de/gstool

2. Project status

   GSTOOL is developed by the German Federal Office for Information Security (BSI) with the purpose to support users in performing analysis and management of IT-risks according to standard BSI safeguards. The last version 4.0 is from 2007, the project is active.

3. License type: free license for the German government at all levels (federal, state and local). Commercial license is also available, as well as 30-day evaluation version with full functionality.

4. General purpose

   The tool is designed as a support for creation, management and update of IT security concepts which conform to the BSI IT-safeguards. These safeguards have been standardized in Germany (http://www.bsi.bund.de/gshb), and are obligatory for all government bodies. After the relevant information have been collected with the help of the tool, it offers the evaluation of IT-security risks for the given (described) infrastructure.

5. Platform: Windows.

6. Model

   (a) Model class: qualitative, process- and IT-level.

   (b) Model type: risk analysis following BSI IT-safeguards (BSI standard 100-3 [38]).

   (c) Model description

   The risk assessment process as prescribed by BSI IT-safeguards comprises the following steps:

   - Preparation: initiation of a systematic IT-security process, IT-structural analysis, specification of safeguard requirements (with respect to security attributes such as trust, integrity and availability), modeling based on IT-safeguard catalog, basic security check and potential security analysis.
   - Compilation of a threat overview: a tabular overview of threats that are relevant for the IT objects specified in the previous step is generated. All objects for which no threat is identified are deleted. All threats are sorted by the objects they may affect. After that, safeguard requirements of each object are derived. This threat overview will be used as a basis for discovery of further threats.
   - Discovering additional threats: for each IT object, it is possible to specify threats which were not categorized beforehand, or which are not included into the BSI IT-safeguard model. In this step additional custom threat analysis may be performed.
   - Threat rating/evaluation: for all threats identified in the previous two steps, it is checked whether and which existing security measures from the BSI IT-safeguard catalog are applicable. The result of this investigation is recorded for each threat. In this step it is identified which objects can be protected using BSI IT-safeguard measures, and for which objects there are remaining risks even after protection measure application. Those will be handled in the following step.

- Risk treatment: in practice, there will almost always be such threats that cannot be addressed using protection measures from the BSI IT-safeguard catalog. Those threats cause further risk for the IT process that is being assessed. For each unaddressed threat from the previous step, the following alternatives are offered: risk reduction through further protection measures, risk reduction through restructuring, or risk transfer.
- Consolidation of the IT security concept: in case that additional protection measures have been introduced to address specific threats, the security concept must be consolidated. All protection measures for each IT object are therefore additionally checked according to following criteria: collaboration, usability and user friendliness, suitability.

(d) Systems modeled: IT systems in administrative sector (primarily different levels of government) and corporate sector.

(e) Model input: description of an IT system according to [38] (steps Preparation and Compilation of a threat overview above) in textual or tabular form.

(f) Model output: tabular risk rating/evaluation for each IT object with respect to trust, integrity and availability; recommended protection measures.

(g) Interfaces: Microsoft Data Access, SQL-Server, text files.

7. Use cases: federal, state and local government in Germany, as the standard risk assessment method.

8. Assumptions and restrictions: N/A.

B.2.7 CALLIO: Secura 17799

1. Source/Reference: [23]

   Web: http://www.callio.com/secura.php

2. Project status: Callio Secura is a commercial tool, which is being actively developed.

3. License type: commercial and evaluation/demonstration license.

4. General purpose

   Callio Secura 17799 is a product from Callio technologies. It is a web based tool with database support that lets the user implement and certify an information security management system (ISMS). It supports the ISO17799 and ISO 27001 (BS 7799-2) standards and can produce documents that are needed for certification. Moreover it provides document management functionality as well as customization of the tool's database. Audits for other standards such as COBIT, HIPAA and Sarbanes & Oxley can be carried out by importing custom questionnaires.

5. Platform: Web client; requires a relational database (MySQL, MS SQL Server), Web server (IIS, Apache), application server (BlueDragon JX Server).

6. Model

   (a) Model class: qualitative, process- and IT-level.

   (b) Model type: risk assessment based on the ISO17799 and ISO 27001 (BS 7799-2) standards.

   (c) Model description

   The tool supports following phases of the risk assessment, mitigation and management, according to the ISO17799 and ISO 27001 (BS 7799-2):

   - Risk identification: risk assessment module enables to identify vulnerabilities/threats, associate them with assets, and manage suggested list of threats.
   - ISO 17799 preliminary diagnostic: questionnaire, initial judgment regarding the state of security.
   - Risk evaluation: risk evaluation and risk calculation are supported.
   - Policy management/audit preparation: security policy can be created using proposed policies and directives.
   - Asset inventory & evaluation: range of examples grouped in categories.
   - Risk treatment: selection of ISO 17799 controls with flexible list of suggested controls. Different scenarios can be created and evaluated.
   - ISMS diagnostic: verification if the ISMS meets the requirements for BS 7799-2 certification.
   - Risk communication: document management and awareness center portal enable reporting and risk communication.

   (d) Systems modeled: global corporate structure (employees, processes, IT).

   (e) Model input: ISMS goal and scope, ISO 17799 compliance report and inventory and evaluation of the assets to be protected.

   (f) Model output: risk analysis (identification and evaluation of threats, vulnerabilities and requirements, risk calculation), risk treatment (risk treatment plan outline), statement of applicability (controls and ISMS), customized security policies (personalized policies and templates).

   (g) Interfaces

   - Input: multi user Web application.
   - Output: relational database (MySQL, MS SQL Server), report generator.

7. Use cases: government agencies, large scale companies, SMEs.

8. Assumptions and restrictions: N/A.

B.2.8 CCN-CERT: PILAR / EAR

1. Source/Reference: N/A.

   Web: http://www.ar-tools.com/en/index.html, http://www.ccn-cert.cni.es

2. Project status

   PILAR is being developed by the Spanish National Security Agency (CCN) since 2004. It is used for the management of government agencies. EAR is a free version of PILAR, and both versions are under active development (last version released in January 2009).

3. License type

   PILAR is used by the Spanish government. EAR offers a public domain license for the read-only mode, and a commercial license for the fully functional write mode. There is also an evaluation license.

4. General purpose

   EAR/PILAR implements and expands Magerit RA/RM Methodology. It is designed to support the risk management process along long periods, providing incremental analysis as the safeguards improve. Its functionalities include:

   - Quantitative and qualitative risk analysis and management
   - Quantitative and qualitative business impact analysis & continuity of operations

5. Platform: Windows/Linux/Unix.

6. Model

   (a) Model class: quantitative-analytical, qualitative; process-level.

   (b) Model type: MAGERIT method (Methodology for Information Systems Risk Analysis and Management), combined with attack trees, audits, data flow diagrams, process charts and boolean functions.

   (c) Model description

   The process supported by the tool comprises following phases:

   - Phase 1 - Assets: assets are the resources in the information system or related to it that are necessary for the organization to operate correctly and achieve the objectives proposed by its management. The essential asset is the information handled by the system, that is the data. Other relevant assets can be services, applications, equipment (hardware), information media, auxiliary equipment, communication networks, installations and persons. Assets may further depend on each other.

- Phase 2 - Threats: the next step is to determine the threats that may affect each asset identified in the previous phase. There is a predefined elements catalog, from which threats can be selected.
- Phase 3 - Determination of the impact: impact is measurement of the damage to an asset arising from the appearance of a threat. By knowing the asset value (in various dimensions) and degradation caused by the threats, their impact to the system can be derived directly. The only consideration required relates to dependencies between assets.
- Phase 4 - Determination of the risk: risk is the measurement of the probable damage to the system. Knowing the impact of threats to the assets, risk can be derived directly simply by taking into account the frequency of occurrence. The risk increases with the impact and with the frequency.
- Phase 5 - Safeguards: safeguards or counter-measures are procedures or technological mechanisms that reduce the risk. There are threats that can be removed simply by suitable organization; others require technical devices (programs or equipment), while others need physical security. Finally, there is the personnel policy. The elements catalog gives a list of suitable safeguards for each asset type.

(d) Systems modeled: enterprise structure and organization, government organizations.

(e) Model input: asset identification, relationships, and value for the organization.

(f) Model output: availability, integrity, confidentiality, authenticity, risk.

(g) Interfaces

- Input: graphical user interface, XML and CSV formats for import.
- Output: graphical user interface, XML and CSV formats for export.

7. Use cases: government agencies, large scale companies, ICT-sector.

8. Assumptions and restrictions: N/A.

B.2.9 C&A Systems Security: COBRA

1. Source/Reference: N/A.

   Web: http://www.riskworld.net/

2. Project status: the tool was developed in the 90s, but is currently (March 2009) under major re-development/enhancement and not presently available for purchase.

3. License type: commercial and evaluation licenses are available.

4. General purpose

   The tool enables security risk assessment to be undertaken by organizations themselves. It evaluates the relative importance of all threats and vulnerabilities, and generates appropriate solutions and recommendations. It automatically links the risks identified with the potential implications for the business unit. Alternatively, a particular area or issue can be examined, without any impact association. COBRA comes equipped with four discrete knowledge bases that can be further customized using the module manager.

5. Platform: N/A.

6. Model

   (a) Model class: qualitative, process-level.

   (b) Model type: knowledge-based risk analysis

   (c) Model description

   COBRA comes equipped with four discrete knowledge bases: the IT Security (or default) knowledge base, the operational risk knowledge base, the 'Quick Risk' or 'high level risk' knowledge base and the e-Security knowledge base. These all serve different functions: the first two provide for comprehensive and detailed risk assessment in their respective domains. The third enables a rapid high level assessment of a whole business system. The last knowledge base was specifically constructed to cover modern network based systems.

   The Risk Consultant knowledge bases cover security threats comprehensively. Each area of potential risk is addressed, often by specific question/knowledge modules. A sample of those included in the IT Security knowledge base for example are:

   - logical access; development
   - system audit; hardware; hazards; system design
   - operations; networks; personnel; physical access
   - change control; system access; contingency
   - security management; security awareness; security administration
   - systems programming; functional control

   (d) Systems modeled: IT-processes and organizations.

   (e) Model input: process description, list of threats and weak points.

   (f) Model output: identification of system threats, vulnerabilities and exposures; measuring the degree of actual risk for each area or aspect of a system, and directly linking this to the potential business impact; solutions and recommendations to reduce the risks; business and technical reports; certification of the ISO-17799 conformance.

(g) Interfaces

- Input: N/A.
- Output: N/A.

7. Use cases: N/A.

8. Assumptions and restrictions: N/A.

B.2.10 DCSSI: EBIOS

1. Source/Reference: N/A.

   Web: http://www.ssi.gouv.fr/en/confidence/ebiospresentation.html

2. Project status: the tool is developed by the Central Information Systems Security Division of the French government. The last available version was released in 2005.

3. License type: open source.

4. General purpose

   EBIOS is a software tool developed by Central Information Systems Security Division (France) in order to support the Ebios method (Expression of Needs and Identification of Security Objectives). The tool helps the user to produce all risk analysis and management steps according to the five EBIOS phases and allows the study results to be recorded and the required summary documents to be produced.

5. Platform: Windows and Linux, PowerPC under Linux, SPARC Architecture under Solaris.

6. Model

   (a) Model class: qualitative, process-level.

   (b) Model type: EBIOS method (Expression of Needs and Identification of Security Objectives).

   (c) Model description

   The EBIOS method comprises following steps:

   - Context study: the purpose of this step is to identify the target system in global terms and position it in its environment so that the target of the security study can be accurately determined. The step consists of sub-steps: study of the organization, study of the target system, and determination of the security study target.
   - Expression of security needs: this step contributes to risk estimation and definition of risk criteria. It also allows system users to express their security needs for the functions and information they handle. The expression of security needs results from the operational requirements of the system, independently

of any technical solution. The step is divided into two activities: production of needs sheets and summary of security needs.

- Threat study: this step contributes to risk assessment. Its purpose is to determine threats affecting the system. These threats are formalized by identifying their components: the attack methods to which the organization is exposed, threat agents that may use them, vulnerabilities exploitable on the system entities and their level. The threats highlighted through this step are specific to the system. Their characterization is independent of the security needs, information processed and functions supported by the system. The threat study includes three activities: study of threat sources, study of vulnerabilities and formalization of threats.
- Identification of security objectives: the purpose of this step is to evaluate and treat the risks affecting the system. The comparison of threats with security needs highlights the risks to be covered by the security objectives. These security objectives constitute the security specifications for the target system and its environment. They must be consistent with all the assumptions, constraints, regulatory references and security rules identified during the study. The level of security objectives and the assurance level must also be determined during this step. The step includes three activities: comparison of the threats with the needs, formalization of security objectives and determination of security levels.
- Determination of security requirements: the purpose of this step is to determine how to achieve the security objectives, i.e. how to treat the risks affecting the system. This requires determining the security functional requirements describing required security behavior and designed to satisfy the security objectives as formulated in the previous step; and the security assurance requirements forming the grounds for confidence that the product or system satisfies its security objectives. These requirements are established on the basis of functional and assurance components proposed by ISO 15408.

(d) Systems modeled: government bodies and enterprise structure.

(e) Model input: Data concerning the organization and its information system (strategic documents, documents concerning the missions, powers and duties, documents concerning the information system, summaries of interviews with the organization's managers); Data concerning the target system; Presentation of the organization; List of general constraints affecting the organization; List of general regulatory references applicable to the organization; Conceptual architecture of the information system; List of essential elements; Functional description of the target system; List of entities; List of threat sources; Entity/element tables; Security needs summary sheet; List

of retained threats; List of assumptions; List of security rules; List of constraints; List of regulatory references; Choice of the security operating mode; Prioritized list of risks; List of security objectives with the strength level.

(f) Model output: a master plan for information systems security, a security policy, an action plan for information systems security, a rational expression of security objectives statement, adapted and justified specifications for prime contracting, a protection profile or security target.

(g) Interfaces

- Input: Java-based, graphical user interface, questionnaire.
- Output: graphical and textual export.

7. Use cases: public and government sector, security consulting.

8. Assumptions and restrictions: N/A.

B.2.11 Fujitsu Interstage Business Process Manager

1. Source/Reference: N/A.

   Web: http://www.fujitsu.com/global/services/software/interstage/bpm/index.html

2. Project status: active and continuous development, as part of the Software AG CentraSite initiative.

3. License type: commercial license with undisclosed price. Evaluation version is also available for download.

4. General purpose

   The tool offers a workflow-based business process manager that encompasses the entire process lifecycle: process modeling, integration, automation, management and optimization. In the context of the availability study, relevant steps are modeling, management and optimization, as they offer a certain degree of availability modeling/assessment capabilities.

5. Platform: Windows, Solaris 9, Red Hat Linux, ES 4.0, HP-UX 11i, IBM AIX 5.3; Directory services: LDAP, Windows Native Directory, Microsoft Active Directory; Databases: Oracle 9i, 10.2, MS SQL Server, Sybase 12.5.3, DB2 8.1

6. Model

   (a) Model class: qualitative, service-level

   (b) Model type: Workflow (BPMN, BPEL, XPDL, Wf-XML)

(c) Model description

The Business Process Manager model tool provides an environment for modeling, testing and refining business process and business rules that govern the processes before the processes (services) are deployed. The model used is a workflow model, which can be specified using any of the following modeling languages: Business Process Modeling Notation (BPMN), Business Process Execution Language (BPEL), XML Process Definition Language (XPDL) or Wf-XML. Defining, refining and maintaining business rules can be collaborative achieved with decision tables.

Processes can be monitored using Business Activity Monitoring (BAM) dashboards. Key performance indicators (KPI) can be defined, as well as rules how to respond when thresholds are met or crossed. Among others, availability metrics can be defined in decision tables, and set up as a KPI. Business Process Manager Analytics can perform process analysis and reporting using predefined metrics. Search, sort and filter capabilities are also included.

The tool offers what-if analysis, that can be used to determine and eliminate process, performance or availability bottlenecks at runtime. For this purpose, audit trails, real-time activity monitoring, and advanced process analysis are provided.

(d) Systems modeled: enterprise and cross-enterprise workflows.

(e) Model input: workflow description.

(f) Model output: business indicators, performance indicators, thresholds and alarms, real-time monitoring.

(g) Interfaces

- Input: Web-based GUI.
- Output: Web-based GUI.

7. Use cases: banking and finance, power industry, travel industry, enterprise software, traffic management.

8. Assumptions and restrictions

Availability is not directly supported, neither in the modeling nor in the analysis part of the tool. Workflow model can be annotated with availability extensions, and metrics can be customly defined in decision tables, providing runtime availability analysis (through KPI mechanism).

B.2.12 HP: Mercury BTO Enterprise Solutions

1. Source/Reference: N/A.

Web: http://www.mercury.com/us/products/

2. Project status: the tool suite is a commercial product of the Hewlett-Packard corporation. It is under active development.

3. License type: commercial license (details unknown) and trial versions (limited to 14 days of usage) are available.

4. General purpose

   Mercury BTO (Business Technology Optimization) Enterprise is a suite of applications (tools) that supports implementation of ITIL service management. The tools are technologically bound by dashboards, a CMDB (Configuration Management DataBase), and a workflow engine that automates and integrates service management processes.

   ITIL defines a wide scope of actions and some of them are unrelated with the availability. Therefore, we present the overall capabilities of Mercury BTO in brief and provide more details on the availability assessment tools. Mercury Project and Portfolio Management Center provides CobiT support (it also provides support for other quality programs).

   BTO consists of four optimization centers and two lifecycle solutions for managing application change and performance:

   - Application Change Lifecycle
   - Application Performance Lifecycle
   - Project and Portfolio Management Center
   - Quality Center
   - Performance Center
   - Business Availability Center

   Such organization is created by the company for easier product positioning on the market of IT governance tools, and there are overlaps between individual sub-suites (for instance, Mercury Universal CMDB is a required part of almost all sub-suites).

5. Platform

   The tool suite operates on Windows family, Linux, zLinux, and UNIX operating systems. Its functionality depends on partial integration with various other software products (such as J2EE application servers, portal solutions, SOA platforms, enterprise databases) and the tool suite supports all major vendors and products in these categories.

6. Model

   (a) Model class: qualitative, monitoring; service- and process-level.

   (b) Model type: ITIL, CobIT, CMDB.

   (c) Model description

   As the goal of the suite is to support implementation of the ITIL service management, it can be said that the wrapping model of the whole suite is ITIL. However, the suite does not follow ITIL specification clearly nor does it provide clear boundaries between various ITIL parts.

Within individual tools, the main goal is usually improvement of manageability of large software products. The model is often just a simple statistical calculation (e.g., mean values over time) or a thin layer used for storing, retrieving and visualizing data.

(d) Systems modeled: IT-processes.

7. Detailed description

The BTO suite covers a wide set of actions required in enterprise project management (including service-desk tasks or financial impact of changes in a project). Here we give more details about the availability-relevant tools only.

Mercury Application Change Lifecycle is used to enforce the application change process within the organization as it was defined by the management. Without dedicated tools, there are notable differences between what should be done and what was actually done regarding the change process. This results in long change cycle times, a broken process, and failure to identify change impacts and collisions before deployment into the production environment. This commonly leads to outages and extended problem resolution times. The tool is used to manage changes and to mitigate business risks throughout the change lifecycle. It aims at automation of the IT Service Management process, reduction of operational costs and minimization of the risk of application downtime. The tool supports:

- Test management: to ensure that each change fulfills quality requirements.
- Change, configuration, and release management helps in assessment of the business impact of every change prior to release to production.
- Change deployment automates the application release management process to ensure that changes are rolled out successfully.
- Change monitoring ensures that all deployed changes are monitored to ensure that business services are not adversely impacted.

Mercury Application Performance Lifecycle manages performance across the application lifecycle. It is closely related with *Mercury Performance Center* which is an enterprise-wide application performance testing platform. Performance Center provides performance validation and diagnostics. The Application Performance Lifecycle additionally supports:

- Integration between Performance Center, Business Availability Center and testing scripts' creation tools.
- Performance monitors (Real User Monitor and Business Process Monitor) provide information on user's experiences, capturing actual error messages and generating scripts for load testing and business process monitoring. They reduce time of problem resolution and accelerate test script creation time. Performance issues can be located and resolved by diagnostics and profiler tools.

Mercury Project and Portfolio Management Center provides information about demands being made by IT departments, IT projects' portfolio, and the deployment of application changes at the enterprise (organization) level. It includes integrated applications to manage demands, portfolios, programs, projects, resources, financial and application changes. It is oriented towards improvement of business value delivered by the IT. It supports quality programs and process control frameworks such as Six-Sigma, PRINCE2, CMMI, and COBIT. Some of its possibilities are:

- IT activities alignment with business goals
- Rapid adaptation to business changes
- Improved personnel management and task allocation
- Financial visibility and governance throughout the IT lifecycle
- Integration with other Mercury products.

Mercury Quality Center is a Web-based system for automated software quality testing and management across different application environments (e.g., J2EE, .NET, Oracle and SAP). Its purpose is to validate both functionality and automated business processes and identify bottlenecks in production.

Mercury Business Availability Center is a top-down approach to integrating business, end-user, and system perspectives, thus providing a picture of the complex infrastructure that underlies key applications. It consists of several sub-applications.

Mercury End User Management proactively monitors Web site and application availability in real time. It proactively emulates end-user business processes and supports various protocols in Web and non-Web environments, and packaged applications (e.g., Oracle, Siebel, SAP, Citrix). This enables easier identification and resolution of performance and availability issues.

Mercury Diagnostics is a solution for composite and traditional applications management in production and pre-production environments. It makes the resolution of various problems that can impact business availability easier, based on the information extracted from real users or scripted transactions. Examples of these problems include: online portal failures, applications running out of memory or node failures in data/ computation clusters. It also provides performance diagnostics, so it belongs to the Mercury Performance Center as well. The tool supports a variety of platforms and solutions, such as: J2EE application servers (Weblogic, Websphere, Oracle, Fujitsu Interstage, TMaxSoft, ATG, and Sun One), portal solutions (Weblogic Portal, Websphere Portal, and SAP Enterprise Portal), SOA platforms, packaged application technologies (SAP, Oracle), mainframe back ends (MQ Series, CICS), enterprise databases (Oracle, SQL Server, DB2), Microsoft .NET Common Language Runtime, Java Virtual Machines (from Sun, IBM, BEA, and HP), open source platforms

and frameworks (JBoss, Tomcat, Struts, and Hibernate). Mercury Diagnostics uses agents that collect performance, availability, and diagnostics data from applications without the need for application's source code modification or recompilation. It uses byte code instrumentation and industry standards for collecting system metrics. The diagnostics data can be exported to XML format.

Mercury Problem Isolation is used for identifying, diagnosing, and resolving problems. It can isolate the problems by identifying relationships and dependencies among the systems and infrastructure that supports them. It can complement the ITIL service delivery process in organizations utilizing the ITIL framework. It acts as a single point of access for problem information and resolution and uses Mercury CMDB (Configuration Management Database) for identifying and presenting configuration items and changes that might be causing the problem. It correlates key performance indicators with configuration items and associated changes in order to estimate the possible problem causes. If integrated with other Business Availability Center applications (e.g., Business Process Monitor, SiteScope, Universal CMDB), it provides detailed data about components related to a problem.

Mercury System Availability Management provides an organization-level infrastructure monitoring solution. System Availability Management can connect to the existing Enterprise Management System (EMS) products or work together with the Mercury SiteScope to collect and monitor system availability and performance data in the organization. System Availability Management is based on an agentless architecture and it enables centralized management and configuration of the infrastructure. The collected data is organized into groups in order to improve readability. System Availability Management can group information per application (such as CPUs, disk space, database indexes, API values, etc.). Data is collected and stored in the Mercury Business Availability Center repository so that information from multiple monitoring sources can be combined.

Mercury Service Level Management proactively manages service levels from the business perspective and provides service level agreement (SLA)-compliance reporting for complex business applications in distributed environments. It provides the mapping between business service level requirements and operational requirements and can create alerts in case of SLA breaches. It provides a bridge between IT-centric SLA metrics (e.g., CPU uptime, database availability) and business availability metrics by comparing actual application performance with business goals. Because of its business-level orientation, the Service Level Management tool is frequently used together with the Project and Portfolio Management Center. Service Level Management enables the user to define his own availability and performance objectives that reflect individual business goals; measure performance and availability as experienced by end users; and isolate and resolve performance problems before service-level objectives are breached.

Mercury Universal CMDB is a configuration management database which is used to capture, document, and store service dependencies and associated infrastructure that support business services. It is capable of auto-discovery of configurations and configuration item dependencies, visualization and mapping of business services, and tracking of configuration changes. It is built on an open architecture (SOAP Web services) and it can link data to and from other repositories. The Universal CMDB provides other capabilities such as impact analysis, access controls, and management services needed to create and maintain the CMDB. It is one of the core components of Mercury Business Availability Center and Mercury Change Control Management, providing support for the ITIL-based Change, Configuration, and Release Management initiatives. The CMDB handles the physical configuration items such as servers, networking and storage devices, software, but also the logical items such as business applications, virtual private networks, end users, and service level agreements. Automated discovery spans network, servers, mainframes, storage, and application software. Via this discovery process, Mercury Universal CMDB stores cross-tier, peer-level, and other complex relationships. Because of preserved history of configuration item changes, it is possible to discover change-induced outages, audit the change management process, or ensure compliance and consistency with enterprise standards. The tool also provides several template correlation rules (they can be also customized) to perform impact analysis specific to their environment.

Mercury Application Mapping provides insight into the dynamic relationships between applications and the underlying infrastructure. It continuously updates and maintains this topology map within a common relationship model, enabling managers to assess the impact of IT issues on business. The tool performs continuous and non-intrusive exploration of every infrastructure asset in the production environment. Discovery patterns direct the exploration of generic element types, such as network equipment and servers, or specific application elements, such as BEA WebLogic, Oracle, or SQL. It determines inter-relationships between enterprise applications and their underlying infrastructure. Mercury Application Mapping can use different mechanisms to automatically explore the infrastructure for only those elements, relationships, and dependencies needed to perform correlation and impact analysis. This capability is enabled by over 130 discovery methods, which cover level 2 to 7 of the standard OSI Model.

Mercury SiteScope is an agent-less monitoring system designed to ensure the availability and performance of distributed IT infrastructures e.g., servers, operating systems, network devices, network services, applications, and application components. A user connects to SiteScope using a Web browser to view status information and make configuration changes. The architecture consists of a set of objects that perform various functions:

- WebPage Objects display the current status information, present forms for editing and updating of the configuration. These objects also enforce access controls, such as user name and password, that restrict who is allowed to access the SiteScope Web pages.
- Scheduler Objects coordinate when monitors are run, alerts are created, and reports are generated.
- Monitor Objects collect information about the system being monitored. There are objects for monitoring application logs, CPU usage, disk space, processes, Web server throughput, DNS servers, mail servers, access to Web pages, and network response. The Monitor API allows custom monitor objects to be added to handle application specific monitoring needs.
- Alert Objects send alerts (e.g., email, SNMP trap messages) about exceptional events.
- Report Objects generate reports summarizing monitoring activity. They read history information from the log files, summarize, filter, and generate HTML reports in graph and table format.

Although it is claimed that the tool is "agentless" (it is not required to install agents or software on production systems) such qualification is partially misleading since it implies the low level of intrusion on the observed system. However, by "agentless" is meant that there is no need to install additional software for communication of collected information and that remote monitoring is performed by logging into systems as an user and then accessing the Monitor API. Actual monitors consume system resources as well as data transfer of measurements to the SiteScope data repository.

8. Use cases: marketing, insurance, air and space, production and logistics, travel.

9. Assumptions and restrictions: N/A.

B.2.13 IBM High Availability Services

1. Source/Reference: N/A.

   Web: http://www-935.ibm.com/services/us/index.wss/offerfamily/bcrs/a1026936

2. Project status: this is a consulting service offered by IBM.

3. License type: N/A, a consultation with IBM has to be requested

4. General purpose

   IBM high availability services help in avoiding downtime and recovery by assessing, planing, designing, building, implementing and managing an infrastructure that supports the ongoing business availability and service

level objectives. IBM high availability services are designed to help enable consistent management of the critical business processes, IT systems and operating environments, and networks. These services are aimed to reduce downtime, decrease infrastructure complexity and enhance the usage of IT resources. The following service (consulting) options are offered:

- High availability assessment
  - Analysis of IT availability plans, processes, procedures, roles, responsibilities, reporting, controls and service level attainment
  - Analysis of any outages that might occur through post incident review, cost of outage or component failure impact analysis techniques
- High availability planning and design
  - Planning for high availability, including plans, program management, reporting and service level management
  - Process and procedure designs for high availability, including roles and responsibilities
  - High availability technology architecture designs for IBM System z, IBM System i, IBM System p, IBM System x and multivendor servers, as well as storage and networks
- High availability implementation
  - Planning, design and implementation for the IBM Server Optimization and Integration Services
- Capacity services for high availability
  - Scheduled capacity services for System z, System i, System p and multivendor servers; IBM and non-IBM storage; and networks
  - Data center and workplace continuity hosting

5. Platform: N/A.

6. Model

   (a) Model class: qualitative, service-level.

   (b) Model type: questionnaire.

   (c) Model description

   Availability assessment within this tool works using qualitative questionnaire method that identifies the following types of threats: business driven threats, data driven threats and event driven threats. The following are some questionnaire examples from all three categories.

   Business driven threats: business continuity management and regulatory compliance fall into the category of business driven threats. The usual question asked are: Do you have a plan that considers how you will manage addressing government and industry regulations locally and across the globe? How prepared you are to mitigate risks

to non-compliance? Whether your business is prepared for success? Can your infrastructure handle spikes in demand?

Data driven threats: these threats involve the corruption, theft and loss of data. Some questions to ask regarding data include: Is your back-up data stored in a place that could be compromised? Have you tested your restoration process to ensure it works? Do you have a back-up plan if your backups are destroyed? Can you produce data on demand in an audit situation? Can your data be accessed by authorized users within a specified time frame?

Event driven threats: these events cannot be predicted but actions can be taken to help prepare for them. These threats range from natural disasters to terrorist actions to pandemics. Critical questions to ask include: How will you deal with employees who cannot, or will not, come to work? Do you have a plan in the event that normal communications are not available? Do you have a plan in the event that there is limited air transportation, fuel or rental cars?

7. Use cases: N/A.

8. Assumptions and restrictions: N/A.

B.2.14 IBM Tivoli Availability Process Manager

1. Source/Reference: N/A.

   Web: http://www-306.ibm.com/software/tivoli/products/availability-process-mgr/

2. Project status: under active development.

3. License type: commercial license, demo available for download; academic license is also available.

4. General purpose

   Tivoli Availability Process Manager enables availability assessment and incident priority assignment according to their impact not only on IT infrastructure, but also on critical business processes. In that way, the tool establishes a connection between the IT and the business layer. It offers availability management (incident management) based on ITIL and determines business impact of a failure.

5. Platform: IBM AIX 5.2, 5.3, RedHat Linux, Windows.

6. Model

   (a) Model class: qualitative, IT- and Service-level

   (b) Model type: Component Failure Impact Analysis (CFIA) and Availability (Incident) Management, as specified by ITIL.

(c) Model description

The model offers two basic capabilities: availability management and determining business impact of resources and incidents on the business process.

Availability Management: Information Technology Infrastructure Library (ITIL) defines the goal of availability management as optimizing the capability of the IT infrastructure and supporting organization to deliver a cost-effective and sustained level of service availability that enables the business to satisfy its objectives. The ITIL definition of availability management covers only service level availability and leaves the infrastructure level availability to a set of best practices covered under service operations. Tivoli Availability Process Manager extends this process by including various ITIL processes that span across ITIL service delivery and ITIL service support. These processes include the following:

- Event management
- Incident management
- Problem management
- Service level management
- Availability management

Tivoli Availability Process Manager provides tasks that help to determine and understand the business impact of incidents or service disruptions. It also provides critical information that helps to classify, prioritize and handle incidents or service disruptions.

The goal of the ITIL incident management is to minimize disruption to the business by restoring service operation to agreed-upon levels as quickly as possible. The ITIL incident management process is divided into seven key parts:

- Incident detection
- Incident classification
- Initial support
- Investigation and diagnosis
- Resolution and recovery
- Incident closure
- Incident monitoring and communication

The focus of Tivoli Availability Process Manager is on the detection, classification, initial support, investigation and diagnosis of incidents.

Determining business impact: Tivoli Availability Process Manager includes a determine business impact (DBI) task that helps to assess and prioritize the impact that one or more resources have on the business. The DBI task helps in prioritizing an incident based on the resource that is reported to have a problem. In this case, the current status of that resource, potential failing components, affected

business systems and other groupings, and the impacted service level agreements (SLA) are important to making that determination. The DBI task also assesses the impact of a change to a resource. In this case, the current status might not be as important, but the relationships to business systems and other groupings, as well as the SLAs that apply to the resource that is scheduled to be changed, are important. Tivoli Availability Process Manager receives information about IT relationships from the CMDB system. Following steps are included in the DBI task: search, assess failing components, assess services impacted, assess SLA/OLA impact, summary.

7. Use cases: automotive, banking, computer services, energy, government, health, media.

8. Assumptions and restrictions: can be used (efficiently and reasonably) only within Tivoli Unified Process (TUP), which leads partially to vendor lock-in with IBM.

B.2.15 Information Governance: PROTEUS

1. Source/Reference: N/A.

   Web: http://www.infogov.co.uk/

2. Project status: PROTEUS was developed in 1995 and is recommended as the tool for automation of the BS-7799 norm of the British Standards Institution. The last release is from 2008.

3. License type: commercial and demonstration licenses are available.

4. General purpose

   Proteus Enterprise is a compliance, information security risk management, and corporate governance tool. It also allows organizations to implement the controls of any standard or regulation, e.g. BS ISO/IEC 17799 and BS ISO/IEC 27001, BS 25999, SOX, CobiT, PCI DSS etc.

5. Platform: Windows and Linux (server); Windows (client).

6. Model

   (a) Model class: qualitative, IT- and process-level.

   (b) Model type: risk analysis and assessment based on the BS-7799 recommendation.

   (c) Model description

   The tool supports the following risk analysis and assessment phases:

   - Asset inventory & evaluation: supported by location but cross-referenced across an entire, multi-national or distributed, organization

- Risk identification: qualitative and quantitative risk assessment techniques supported, integrated with asset management, threats, countermeasures, risk treatment plans and incident management
- Risk analysis: relative and absolute risk scales can be used to adapt to corporate risk thresholds
- Risk assessment: 5 stage generic process, which can be mapped to BS ISO 27001, IRAM or other methodologies
- Risk evaluation: 5 types are supported (physical, information, service, application and group), threats can be automatically inherited via asset relationships, location and asset profile
- Risk treatment: action plans are integrated with compliance, risk assessment, business impact analysis, business continuity and incident management
- Risk acceptance: audit trail of system changes
- Risk communication: every aspect of the system can be reported or viewed by secure PDFs, customizable business objects reporting, and via the optional Proteus RiskView management information graphical dashboard

(d) Systems modeled: IT companies and processes.

(e) Model input: process description with asset inventory.

(f) Model output: business impact measures, risk analysis, action plans.

(g) Interfaces

- Input: Web browser.
- Output: Web browser.

7. Use cases: finance, telecommunications, pharmaceutical, retail, government.

8. Assumptions and restrictions: N/A.

B.2.16 Insight Consulting/Siemens: CRAMM

1. Source/Reference: N/A.

   Web: http://www.cramm.com/

2. Project status: the first version of CRAMM (CCTA Risk Assessment and Management Method) was developed in 1985, on request of the British government. After the initial success, the commercial and publicly available version was released. In the meantime, Insight has been bought by Siemens, which is developing the tool further. The last known release is 5.2 from 2009.

3. License type: commercial as well as 30-day evaluation license.

4. General purpose

CRAMM offers a range of risk assessment tools that are compliant with ISO 27001 and address tasks such as asset dependency modeling, business impact assessment, identifying and assessing threats and vulnerabilities, assessing levels of risk and identifying required and justified controls on the basis of the risk assessment.

5. Platform: Windows.
6. Model
   (a) Model class: qualitative, process-level.
   (b) Model type: risk analysis and assessment following the CRAMM, compliant with ISO 27001 and BS-7799 standards.
   (c) Model description
   CRAMM risk assessment tool can be used to answer single questions, to look at organizations, processes, applications and systems or to investigate complete infrastructures or organizations. Users have the option of a rapid risk assessment or a full, more rigorous, analysis. The following risk aspects can be answered:
      - Determining if there is a requirement for specific controls, e.g., strong authentication, encryption, power protection or hardware redundancy
      - Identifying the security functionality required for a new application
      - Developing the security requirements for an outsourcing or managed service agreement
      - Reviewing the requirements for physical and environmental security at a new site
      - Examining the implications of allowing users to connect to the Internet
      - Demonstrating compliance with legislation such as the Data Protection Act
      - Developing a security policy for a new system
      - Auditing the suitability and status of security controls on an existing system
      - ISO 27001 compliancy
   (d) Systems modeled: IT-processes and infrastructures.
   (e) Model input: process description with asset lists.
   (f) Model output: risk assessment, countermeasures and suggestions.
   (g) Interfaces
      - Input: tables and questionnaires.
      - Output: Microsoft Office (Word, Excel), graph, text.
7. Use cases: government (UK), defense (NATO), aerospace (BAE Systems, Royal Air Force), IT (IBM), automotive (General Motors), banking and finance (Swiss Bank), telecommunications(T-Mobile).

8. Assumptions and restrictions: N/A.

B.2.17 RiskWatch: RiskWatch for Information Systems

1. Source/Reference: N/A.

   Web: http://www.riskwatch.com

2. Project status: the tool is under active development, as part of the RiskWatch tool suite. Last version of the RiskWatch for Information Systems is from 2007.

3. License type: commercial license, on-line demonstration.

4. General purpose

   This tool conducts automated risk analysis and vulnerability assessments of information systems. The knowledge databases that are provided are customizable, including the ability to create new asset categories, threat categories, vulnerability categories, safeguards, question categories, and question sets. The tool includes controls from the ISO 17799 and US-NIST 800-26 standards as well as support for ISO 27001, COBIT 4.0 and Sarbanes Oxley (SOX).

5. Platform: Windows.

6. Model

   (a) Model class: qualitative, IT- and process-level.

   (b) Model type: risk analysis and assessment based on the ISO 17799, ISO 27001, COBIT 4.0 and Sarbanes Oxley (SOX).

   (c) Model description

   The RiskWatch analysis engine analyzes the relationship between the asset values, potential loss impacts, threats and vulnerability categories to illustrate the current organizational security situation and to automatically recommend mitigating, cost-effective solutions. RiskWatch further automatically develops asset profiles and provides financial values for organizations using an asset configuration wizard based on capital expenditures allocation tables

   RiskWatch analyzes all data, and creates management reports detailing compliance vs. non-compliance, backed up with a set of working papers. Return on investment is calculated for each safeguard and a case summary report is generated to show compliance vs. non-compliance, protection levels, annual loss expectancy data by asset category, threat or loss impact category.

   (d) Systems modeled: IT-systems.

   (e) Model input: process description, asset description (automated with asset configuration tool).

(f) Model output: risk assessment, safeguard list, return on investment analysis, annual loss expectancy, threat impact.

(g) Interfaces

- Input: RiskWatch for Information Systems includes a web-based application that allows employees to answer questions over the Web. Questions are created from ISO 17799, ISO 27001, COBIT 4.0 and SOX standards and are separated by job category and by vulnerability category. Analysts can add, delete or modify questions.
- Output: graphic, export to Office, database.

7. Use cases: government (US Department of Justice, National Security Agency), academia, IT (IBM), telecommunications (AT &T, Verizon, Vodafone).

8. Assumptions and restrictions: N/A.

B.2.18 Self assessment programs by itSMF International

1. Source/Reference: N/A.

   Web: http://www.itsmfi.org/

2. Project status: active.

3. License type: free, the tool is available online or as a set of Excel spreadsheets.

4. General purpose

   The aim of the questionnaire is to give the organization (and its management) an idea how well it is performing compared to ITIL best practice. The questionnaire cannot be used for the purpose of ITIL conformance testing, but it should create the awareness of what may be addressed in order to improve the overall process capability.

5. Platform: online/Microsoft Office.

6. Model

   (a) Model class: qualitative, process-level.

   (b) Model type: ITIL.

   (c) Model description

   The self-assessment scheme is composed of a simple questionnaire which enables the user to ascertain which areas should be addressed in order to improve the overall process availability. The questions are YES/NO, with weight associated with each YES answer and zero value for NO, and based on the answers is determined whether an organization implements the guidance in a particular category. In order to pass the test within a level in a category, all mandatory

questions must be answered with YES and additionally at least one of non-mandatory questions.

Results of the self-evaluation can be submitted to the itSMF for statistical analysis. The analysis is anonymous and it is possible to view the results averaged over all organizations that have already participated in the assessment. This is particularly useful as it can be used for comparison of obtained results with results made by other organizations and companies.

For each of categories (service level management, financial management, capacity management, continuity management, availability management, service desk, incident management, problem management, configuration management, change management, release management), questions are grouped in following levels:

- Level 1: Prerequisites, ascertains whether the minimum level of prerequisite items are available to support the process activities.
- Level 1.5: Management intent, establishes whether there are organizational policy statements, business objectives (or similar evidence of intent) providing both purpose and guidance in the transformation or use of the prerequisite items.
- Level 2: Process capability, examines the activities being carried out. The questions are aimed at identifying whether a minimum set of activities are being performed.
- Level 2.5: Internal integration ascertains whether the activities are integrated sufficiently in order to fulfill the process intent.
- Level 3: Products, examines the actual output of the process to enquire whether all the relevant products are being produced.
- Level 3.5: Quality control is concerned with the review and verification of the process output to ensure that it is in keeping with the quality intent.
- Level 4: Management information is concerned with the process governance and ensuring that there is adequate and timely information produced from the process in order to support necessary management decisions.
- Level 4.5: External integration examines whether all the external interfaces and relationships between discrete process and other processes have been established within the organization. At this level use of the full ITIL terminology may be expected for IT service management.
- Level 5: Customer Interface, is primarily concerned with the ongoing external review and validation of the process to ensure that it remains optimized towards meeting the needs of the customer.

(d) Systems modeled:

(e) Model input: ITIL-questionnaire.

(f) Model output: level of ITIL compliance.

(g) Interfaces

- Input: Web browser.
- Output: Web browser.

7. Use cases

   The analysis results available at the site are anonymous, so it is not possible to obtain participant names. However, most of them come from the IT consulting sector (24.2%), followed by the government organizations (18%). Not every organization participated in each questionnaire, so the numbers vary. For availability management assessment, 677 organizations have used the tool, and for service level management impressive 7415.

8. Assumptions and restrictions: N/A.

B.2.19 Software AG CentraSite

1. Source/Reference: N/A.

   Web: http://www.softwareag.com/Corporate/products/centrasite/default.asp

2. Project status: commercial tool under continuous development.

3. License type: commercial license with undisclosed price, demo version available for download.

4. General purpose

   Centrasite is platform for Service Oriented Architecture (SOA) governance. SOA governance covers two aspects: 1) service implementation and 2) IT governance at the business process level. Governance priorities are performance, risk management, service availability and aligning IT infrastructure with business goals (processes). Centrasite supports governance in an iterative process that includes tracking compliance with service policies, monitoring and managing service levels and availability, and optimization of processes created from services (and vice versa).

5. Platform: platform-independent (Java + browser-based GUI).

6. Model

   (a) Model class: qualitative, service-level.

   (b) Model type: metadata repository supporting policy, change control, maintenance, automation, dependency analysis and reporting at the business process (service) level.

   (c) Model description

   The SOA governance model ensures that IT activities, priorities and decisions align with business goals, and that IT is delivering the value to the business. Governance model encompasses applying policies for mitigating risk, implementing consistent processes and instituting common metrics for tracking performance, quality of service and

dependability. In the context of our study, the last property is of extreme importance. For example, successful governance model must ensure that services will not jeopardize the performance or availability of internal back end system effectively connecting business process (service) and IT layer. SOA governance model involves managing services across the entire service lifecycle, from design to run time.

Design time governance specifies ground rules for service creation, architectural standards and service contracts. Run time governance involves verifying compliance with contract and policies, performance, SLAs, QoS (e.g., availability, dependability, security). Governance model also includes a change phase, which requires enforcement of metapolicies and conducting impact analysis of service changes on IT operations, system/data interfaces, contract compliance.

CentraSite is based on the repository governance model, where repository stores metadata about the services, providing the abstraction layer between services and implementations. Concretely, repository stores models, mappings, shared keys, transformations, process models, business rules, test plans, runtime performance histories etc.

Centrasite includes several tools, some of which are relevant to the objectives of this study. Those tools will be described in more detail, others will be only mentioned for the reason of completeness.

Crossvision Information Integrator is a tool for organizing, managing, aggregating, transforming and filtering business data. *Crossvision Service Orchestrator* is a tool for Web Services and XML-based message routing, schema validation and transformation and sequence workflows.

Ilog manages business decisions through rules-based decision services. In our context, it is interesting that rules can be based on QoS metrics (such as availability). *Mega* tool provides reporting and analytic information in the process of associating services and business goals. It is possible to measure the alignment of selected services to the organization's business processes, or gauge the level of availability and dependencies between services and business processes. *Parasoft* tool provides test assets, such as reusable tests, regression test, emulated services, functional, performance and availability tests. *AmberPoint* tool ensures compliance at runtime with specifications made at design time. It can flag the following exceptions: dependencies, rogue services, dynamic changes to the service network, service levels, security, auditing, logging, debugging and failures. Policies serve to control runtime behavior to ensure high availability, performance and security. These policies are enforced automatically.

7. Use cases: all community members are also applying Centrasite, among others Fujitsu, Novell and Software AG. Completed projects using Centrasite can be found at http://www.softwareag.com/Corporate/Solutions/

Customers/References/default.asp, among others Belgian National Railway Company, Commerzbank, DaimlerChrysler, Nissan Motors, Scandinvian Airlines, Vodafone, Volkswagen, ZDF, etc.

8. Assumptions and restrictions

   CentraSite is based exclusively on Web Services (SOAP, WSDL, UDDI). Furthermore, it offers proprietary metadata model and availability policies and metrics, which could be difficult to integrate.

B.2.20 Telindus Consultants Enterprises: ISAMM

1. Source/Reference: N/A.

   Web: http://www.telindus.com/

2. Project status: commercial product under active development, last version is from 2008.

3. License type: commercial license as well as trial/evaluation version.

4. General purpose

   ISAMM or Information Security Assessment and Monitoring Method tool follows the set of controls of best practices in information security from the ISO/IEC 27002. ISAMM risk assessment contains three main parts: scoping, assessment and reporting.

5. Platform: N/A.

6. Model

   (a) Model class: qualitative, process-level.

   (b) Model type: risk analysis and assessment following the ISAMM method.

   (c) Model description

   The method comprises three phases:

   - Scoping: as required by ISO/IEC 27001, the first step of a risk assessment is to select the relevant pre-defined asset types and to define the most important assets of each of these types. The selected assets must be valuated for replacement, confidentiality, integrity and availability cost. Second step is to select the relevant threats amongst 15 pre-defined generic threats. Based on these selections, ISAMM will consider the relevant mappings between the chosen asset types and threats in order to generate a number of appropriate threat scenarios. ISAMM will then list all ISO/IEC 27002 controls that could have an effect on these risk scenarios. Optionally, the respondent can indicate a mandatory status for a control. One should therefore assess the existence of mandatory company policies and/or legal and regulatory requirements (e.g. FDA, Sarbanes-Oxley, local laws).

- Assessment: in this phase, the respondent has to complete the actual compliance level for each relevant ISO/IEC 27002 control. As an option, the respondent has a number of detailed compliance questions for each of the controls providing additional insights and details about the actual vulnerability level. When not completely compliant, the respondent has to provide an estimation of the additional yearly cost to become fully compliant. Further questions are prompted in order to define the threat motivation and the number of exposure points for the threats. Based on this information ISAMM is able to calculate the default threat probability and impact for each of the threat scenarios and to determine the actual risk level for each threat.
- Reporting: using the risk reducing characteristics of each control on each of the threat scenarios, ISAMM is also able to simulate the risk reducing effect of each control improvement and select the most appropriate ones, step by step. In this way an optimal risk treatment plan with resulting residual risk can be derived. After completion of the input and calculations, ISAMM will generate a variety of graphs and tables listing all relevant information.

(d) Systems modeled: IT organizations and processes.

(e) Model input: list of evaluated assets, according to the various cost models; list of relevant threats.

(f) Model output: threat probability, impact, risk level, risk treatment plan.

(g) Interfaces

- Input: N/A.
- Output: graphical, export to MS Office.

7. Use cases: N/A.

8. Assumptions and restrictions: N/A.